Thursday, Sept 22 Happy Autumn!

1. Welcome!
2. Tea and Talk
3. Math Club
4. Homework 2
   ↳ going to try to get hw 1 done/back tonight, if you want to wait until Friday for submission
5. Questions?
6. Alternative solvers, IP.
7. Outro
   ↳ small work

Last time we developed the simplex method, which allows us to solve linear programs. Algebraically, we pivot through the vertices by exchanging out basic variables one at a time in a way that improves the objective function. Geometrically, we're traversing the polyhedron around the vertices via the edges. Simplex can also be used to find if our system is infeasible (by creating the auxiliary program) or unbounded (by finding a variable to pivot that's unrestricted).

So this seems ideal! It's what we wanted several days ago: a single method that checked for infeasibility, unboundedness, or solved. But, is it *good*?

Question: What do we mean by good?

is it fast? → # of steps.

maybe be even polynomial!

can we bound the # of steps based on input size? (m,n)

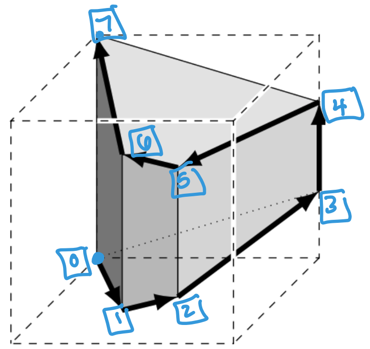
Good news: typically, it's good!

Bad news: when it's bad, it's really bad. Enter, Klee-Minty cubes. Consider the linear program below.

$$(P)\begin{cases}\text{maximize} & 100x_1 + 10x_2 + x_3 \\ \text{subject to} & \begin{array}{rcrcrcll} x_1 & & & & & \leq & 1 & (A) \\ 20x_1 & + & x_2 & & & \leq & 100 & (B) \\ 200x_1 & + & 20x_2 & + & x_3 & \leq & 10000 & (C) \\ x_1 & & & & & \geq & 0 & (D) \\ & & x_2 & & & \geq & 0 & (E) \\ & & & & x_3 & \geq & 0 & (F) \end{array}\end{cases}$$

We can walk through the steps of the simplex method in a very abbreviated form.

step	x_1	x_2	x_3	objective	tight constraints
0	0	0	0	0	(D), (E), (F)
1	1	0	0	100	(A), (E), (F)
2	1	80	0	900	(A), (B), (F)
3	0	100	0	1000	(D), (B), (F)
4	0	100	8000	9000	(D), (B), (C)
5	1	80	8200	9100	(A), (B), (C)
6	1	0	9800	9900	A E C
7	0	0	10000	10000	C D E



The sequence of basic feasible solutions from the simplex method corresponds to a trip in the cube like in the sketch above. Note that it is indeed just a sketch since not all coordinates are same length.

Question: Can you generalize this program to n dimensions? Do you see a problem forming?

max $10^{n-1}x_1 + 10^{n-2}x_2 \cdots x_n$

$x_i \geq 0$

bounds extended similarly.

$2^n - 1$

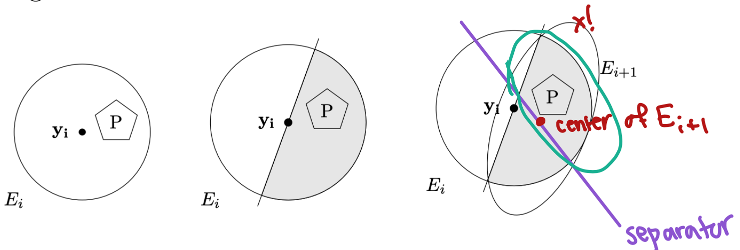
So in general, simplex works well until it doesn't. As such, explorations into new methods for solving linear programs have been needed. We'll briefly talk about two, both of which could make interesting capstone explorations! It's also worth mentioning that these are highly generalizable, and are often stated in terms of *convex optimization.*

Ellipsoid Method: the first algorithm to solve a linear program in polynomial time.

Big idea: we can reduce any linear program to finding a point in a polytope. To find this point, we essentially "zero in" on it!

(1) $E_1 := B(0, R)$, $i := 1$
(2) if the center y_i of E_i is in P, point found!
(3) otherwise, there is a separating hyperplane cutting out half of E_i
(4) pick E_{i+1} to be the smallest ellipsoid containing the half of E_i that contains P
(5) $i := i + 1$ and goto 2.

$x^T A x \leq 1$



But: while mathematically clever, it runs slowly. It did give rise to the next idea!

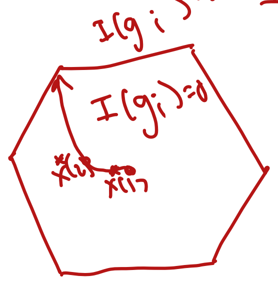
Interior Point Method: polynomial time and simplex competitive!

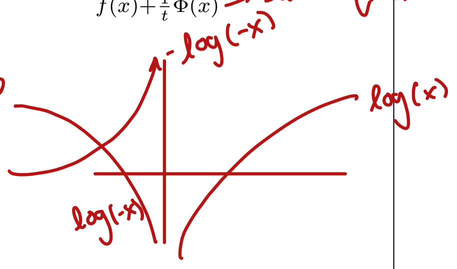
Big idea: start at the analytic center of your polytope. Move along a central path that increases the objective function until you hit boundary.

x! "remove constraints" approx I w/ log

$$\begin{cases} \min & f(x) \\ \text{s.t.} & g_i(x) \leq 0 \ \forall i \end{cases} \to \min f(x) + \sum_i I(g_i(x)) \to \underbrace{\min f(x) - \frac{1}{t} \sum_i \log(-g_i(x))}_{f(x)+\frac{1}{t}\Phi(x)} \to \text{barier fnctn}$$

$$I(g(x)) = \begin{cases} 0 & g(x) \leq 0 \\ \infty & g(x) > 0 \end{cases}$$





Analytic center of the region: x such that $\Phi(x)$ is minimized! We make a set of problems, P_t for $t > 0$.

$$P_t = \min t f(x) + \Phi(x)$$

which has a unique optimal solution $x^*(t)$. These x^* trace a path from the analytic center to the optimal for the system!

We're closing out our discussion on how to solve linear programs with a return to a problem we saw very early on: the integer program. An integer program is a linear program where at least one decision variable is required to be an integer. We'll set a few up before talking about how to solve them.

Example: Suppose we have a company that has a total of J jobs that need completing and a total of W workers to do them. However, not every worker does every job equally well. Specifically it takes t_{wj} for worker w to complete job j. Write a program where each job is assigned to exactly one employee, each employee gets one job, and we minimize time spent.

decision var: $x_{wj} \to$ does worker w work job j? $x_{wj} \in \{0,1\}$

$$\begin{cases} \min & \sum_{j,w} t_{w,j} x_{w,j} \\ \text{st} & \sum_j x_{wj} = 1 \quad \forall w \leftarrow \text{each worker gets } \underline{\text{one}} \text{ job} \\ & \sum_w x_{wj} = 1 \quad \forall j \leftarrow \text{each job gets } \underline{\text{one}} \text{ worker} \end{cases}$$

$x_{wj} \in \{0,1\}$ $\quad x \in \mathbb{Z}, 0 \le x \le 1$

Example: Suppose we are shipping goods via five different types of containers. Each container has an associated weight and value, shown in the table below. We must meet the following conditions. (1) we cannot send more than 10 containers (2) we cannot send more than 10000 kg (3) if we send type 4, we must send at least one type 3 and (4) we must send at least three of types 1 and 2 or types 3 and 4.

Maximize the value we can send.

Type	1	2	3	4	5
Weight (kg)	30	20	30	70	40
Value ($)	60	70	40	90	50

decision: $x_i \to$ # we send of type i $\quad (x_i \in [0,10], \in \mathbb{Z})$

max $60x_1 + 70x_2 + 40x_3 + 90x_4 + 50x_5$

s.t. $30x_1 + 20x_2 + 30x_3 + 70x_4 + 40x_5 \le 10000$

$[\, x_4 \le 10x_3$ (3)

$$\left[\begin{array}{l} x_1 + x_2 \ge 3y \\ x_3 + x_4 \ge 3(1-y) \\ y \in \{0,1\} \end{array}\right.$$ (4)

$x_i \in \{0,1,\ldots 10\}$

$\sum x_i \le 10$

Example: Suppose you are given an $n \times n$ chessboard, on which we want to place n nonattacking rooks (rooks can attack in any row or column). Write a linear *feasibility* program (no objective function).

$x_{i,j}$: is there a rook on i,j? $x_{ij} \in \{0,1\}$

$$\begin{cases} \sum_j x_{ij} = 1 \quad \forall i & \text{1 in each row} \\ \sum_i x_{ij} = 1 \quad \forall j & \text{1 in each col.} \end{cases}$$

Example: Recall that a *matching* in a graph is a pairing of the vertices such that every vertex is matched with exactly one of its neighbors. Write a linear feasibility program that will find a matching in a graph with n vertices (assume n is even).

Follow up: Assume each edge has a cost, and we want to minimize the total of the matching. How does your program change?

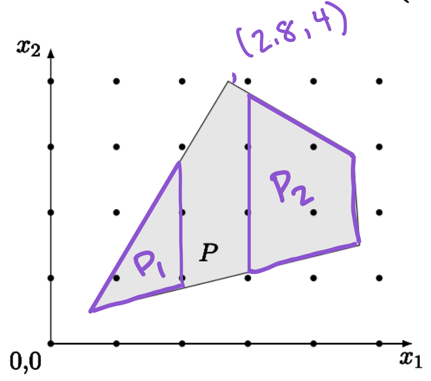
So how do we solve these sorts of things? Is it enough to use a linear program approximation and round? Well, not necessarily. The good news is that we *can* use the simplex method, we just need to adjust it. Our two options are:

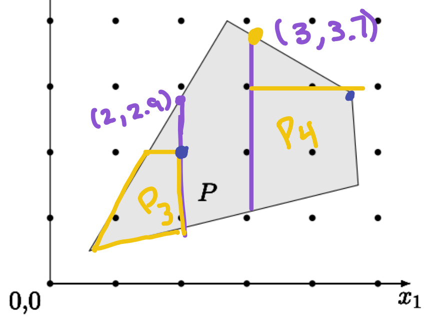
- Branch and Bound: Solve the associated linear program. Then, if one of the decision variables is not an integer ($x_i = a$), create two new programs, each with one new constraint: $x_i \leq \lfloor a \rfloor$ or $x_i \geq \lceil a \rceil$. Solve each with the same rule. This creates a branch and bound *tree*.
- Cutting planes: if our feasible region is a polytope P, cut pieces of it away that don't contain any integer points by making better inequalities. We exploit the fact that if $x_i \leq a$ and $x_i \in \mathbb{Z}$, then $x_i \leq \lfloor a \rfloor$. We then solve the linear program on this new, smaller polytope.

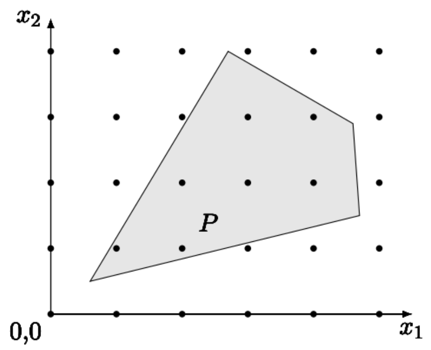
We'll do examples of both of these ideas.

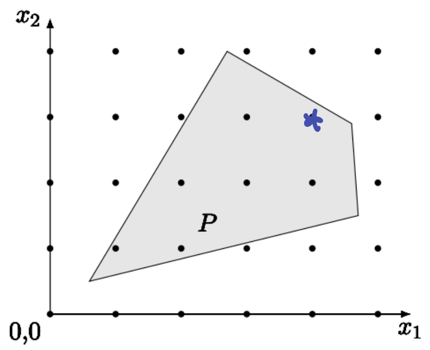
Example: Create the branch and bound tree for the following integer program.

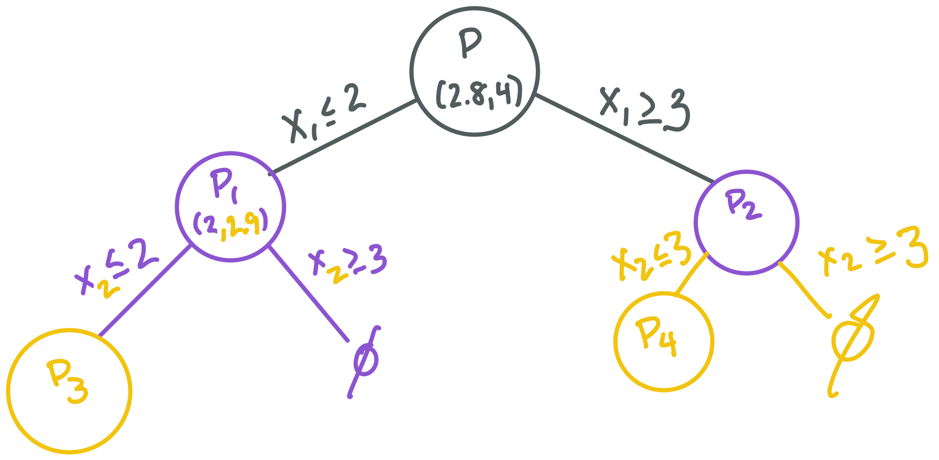
$$IP \begin{cases} \max & x_1 + 100x_2 \\ \text{s. t.} & (x_1, x_2) \in P \\ & x_i \in \mathbb{Z} \end{cases}$$





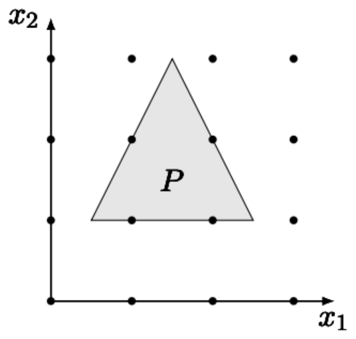






Example: Find a smaller feasible region than the one shown below by manipulating the bounds. Try to do this process twice.

$$\begin{cases} 2x_1 + x_2 & \leq 6 \\ -2x_1 + x_2 & \leq 0 \\ -x_1 & \leq -1 \end{cases}$$



Follow up: In general, if we do this enough times, what shape are we looking at?