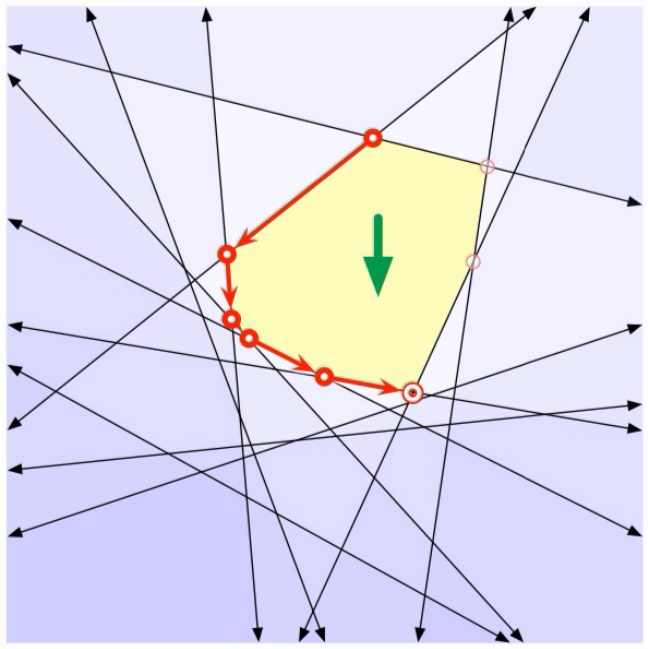
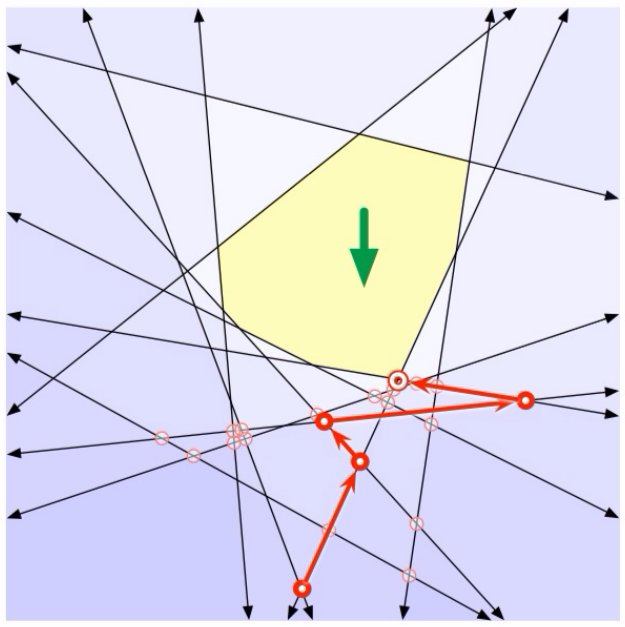
LECTURE 19 (NOVEMBER 6th)

Ellipsoid Algorithm

Last time we saw the simplex algorithm for solving linear programs. We find some vertex of the feasible region and move to a lower vertex in each step.



Primal Simplex



Dual Simplex

Primal (Π)

$$\begin{aligned} \max \quad & c \cdot x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

$\Longleftrightarrow$

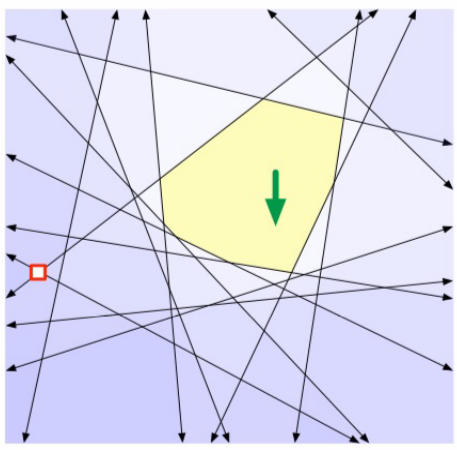
Dual (Ш)

$$\begin{aligned} \min \quad & y \cdot b \\ \text{s.t.} \quad & yA \geq c \\ & y \geq 0 \end{aligned}$$

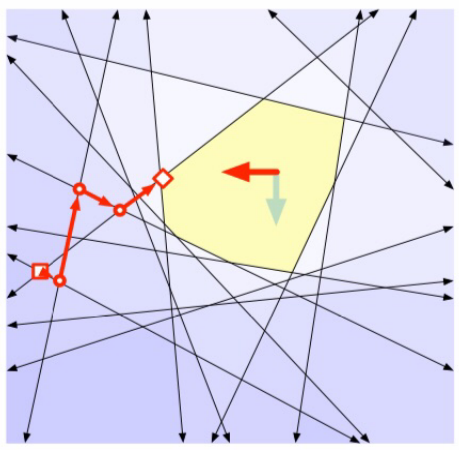
Polytope does not depend on c

Polytope does not depend on b

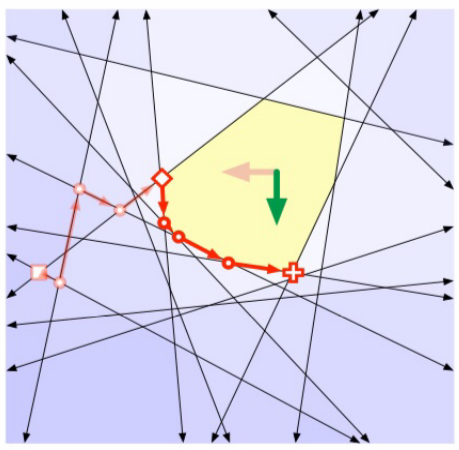
To find the initial point:



① Choose any basis

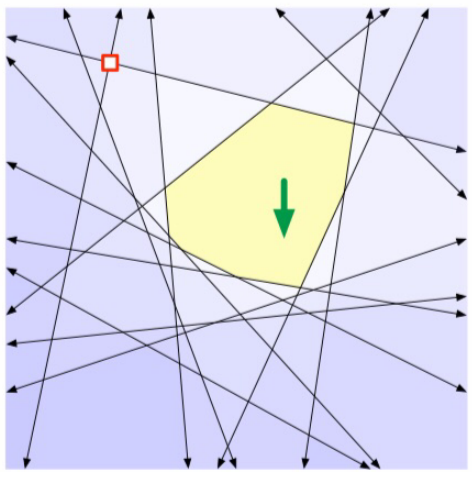


② Rotate the above basis to make it locally optimal and pivot up to a feasible basis

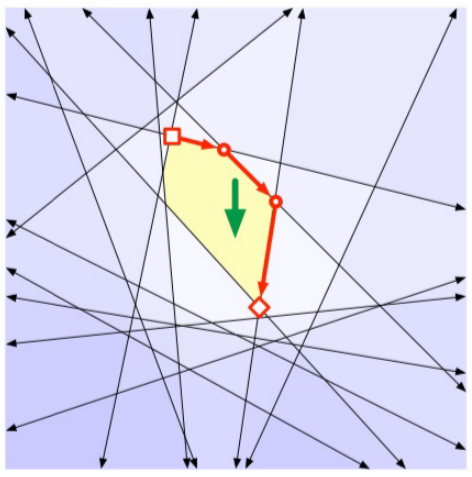


③ Pivot down to a feasible basis with the original objective

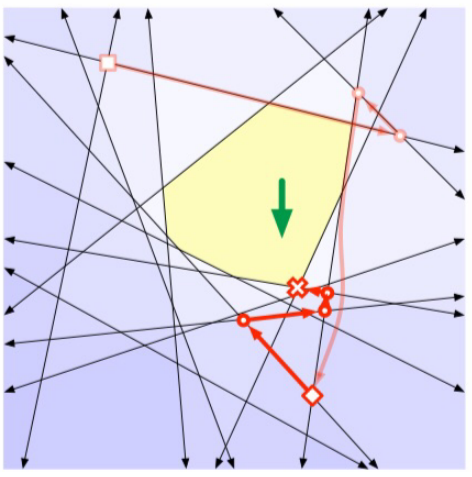
We can view this whole thing through the lens of duality again using the dictionary



① Choose any basis



② Change the offset and find a locally optimal basis



③ Pivot up to the optimal bases with the original offset

This algorithms runs well in practice but there can be pathological cases where this takes exponential time. In fact, there is no known version of this algorithm which does not have a pathological case. In fact, moving to a neighboring vertex at random could still require sub-exponential time.

So, why does the algorithm work well in practice?

Smoothed Analysis — introduced by Spielman-Teng '02

One way to explain this is by smoothed analysis — basically the pathological cases become polynomial time if one adds a small amount of noise to the data and one can prove that for some reasonable noise models, this is always the case and the simplex algorithm runs in polynomial time. Note that this is not worst-case or average-case analysis. We are taking a worst-case input and adding a small amount of noise and analyzing the behavior of the algorithm on this perturbed input.

However, whether this explanation is adequate or not is open to interpretation and this is an active area of research with many longstanding open problems.

In fact, a more basic question remains open:

Polynomial Hirsch Conjecture

Let G be the graph of an n-dimensional polytope with m facets.
Then, the diameter of G is polynomial in n and m.

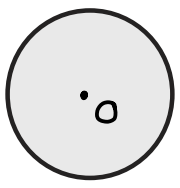
Ellipsoid Algorithm

Today we are going to see an algorithm that we can prove always runs in polynomial time. This is the Ellipsoid Algorithm by Khachiyan from 1979. This algorithm, however, is not very practical since the exponent of the polynomial and the constants are large.

Later in 1980s, an algorithm was found by Karmakar that was both theoretically polynomial time and also works well in practice. This algorithm is based on interior point methods that we will not discuss here in this course.

Ellipsoid algorithm, as the name suggests works with Ellipsoids, so let's see how to describe them mathematically.

A ball is described by the region $x^2+y^2 \leq 1$
This is a ball centered at 0 and with radius 1.
This can also be generalized to higher dimensions

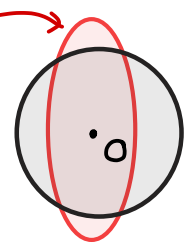


$$x_1^2 + x_2^2 + \cdots + x_n^2 \leq 1$$

An ellipsoid is a squished ball and looks like

$$\frac{x_1^2}{a_1^2} + \cdots + \frac{x_n^2}{a_n^2} \leq 1$$

For example, $(2x)^2 + \left(\frac{y}{2}\right)^2 \leq 1$



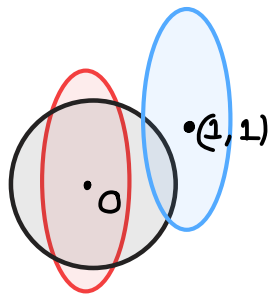
In 2-dim, area of an ellipsoid is
$\pi a_1 a_2$

so, the ratio $\frac{\text{volume (ellipsoid)}}{\text{volume (ball)}} = a_1 a_2$

In d dimensions, the ratio is $a_1 a_2 \cdots a_d$.

One also center the ellipsoid at a different point,
For example,

$$(2(x-1))^2 + \left(\frac{y-1}{2}\right)^2 \leq 1$$



is centered at $(1,1)$. This does not change the volume of the ellipsoid.

Also, note that applying a rotation also does not change the volume.

With the above setup, let us suppose we have an LP

$$\begin{aligned} \max \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where A, c, b are integers and total # bits to describe them is T

We will give an algorithm that runs in time $poly(R)$

In order to do this, we first need some basic facts

① **FACT** If the LP solution is finite, then the optimal objective value is at most $2^{O(poly(T))}$

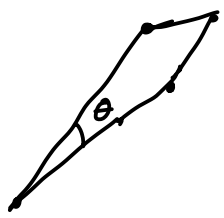
Proof (sketch) If LP solution is finite, it is at a vertex x^*.
One can see that every vertex satisfies

$$Mx = s \quad \text{for some } d \times d \text{ matrix } M \text{ (where } n < d\text{)}$$
$$\text{and length } d \text{ vector } s$$

The solution x is given by $x = M^{-1}s$ and the size of numbers in B^{-1} is polynomial in T.

② **FACT** If there is a finite solution, then volume of the feasible region is atleast $2^{-poly(T)}$

Proof (Sketch) The smallest angle can be generated is $2^{-poly(T)}$



□

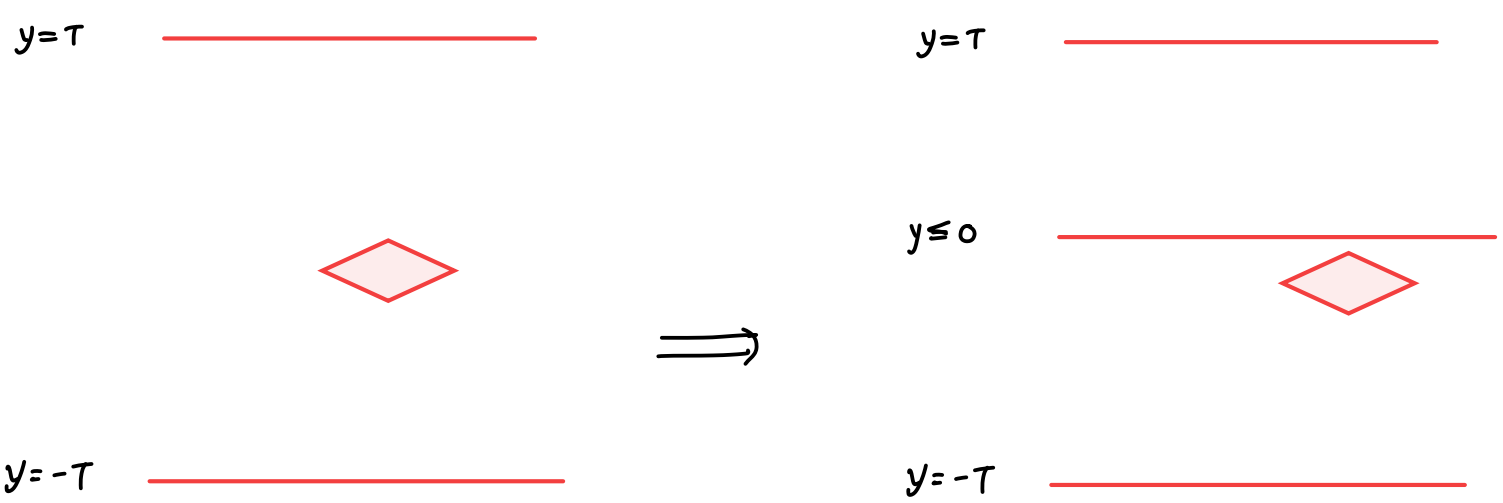
The ellipsoid method reduces the task of solving the LP to just deciding if the LP is feasible or not and to find a feasible point if it is.

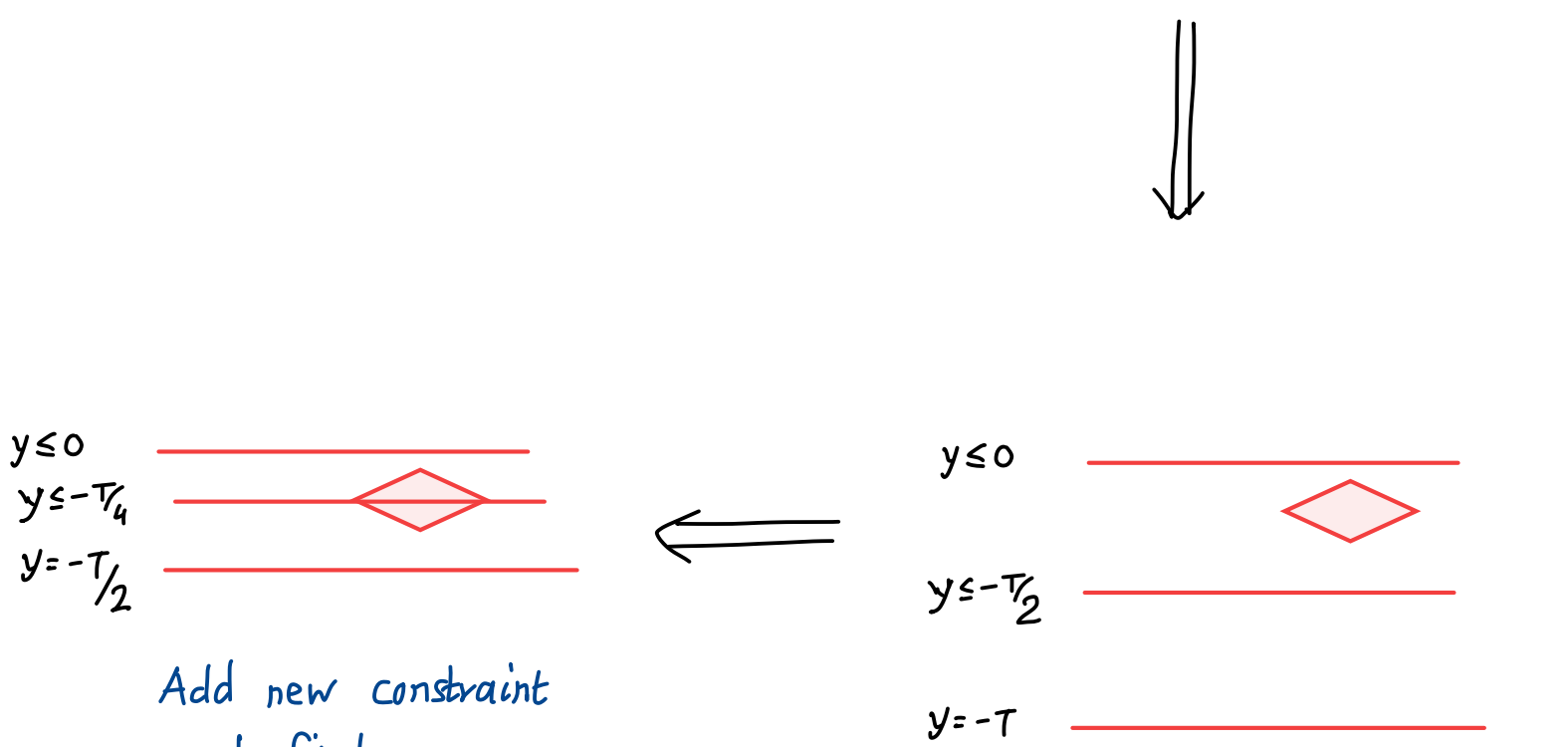
This is done via

$$\max\ c^T x \quad \text{s.t.}\ Ax \le b,\ x \ge 0 \quad \Longrightarrow \quad \text{Choose a number } t:\ \text{Is there an } x \text{ s.t. } c^T x \ge t,\ Ax \le b,\ x \ge 0$$

If we can find x inside the new feasible region in poly time, we can use binary search to find the best value of t in poly time!

From FACT 1, we know that the optimal value $c^T x$ is always in the interval $[-2^{poly(T)}, 2^{poly(T)}]$, so that t always lies there. Now, this is a real line but from FACT 2, we know that we do not need granularity smaller than $2^{-poly(T)}$. Overall, it means that we only need poly(T) rounds of binary search to find the optimal solution.

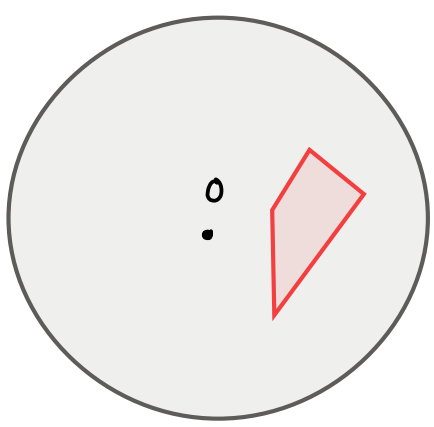




CONCLUSION It is enough to give an algorithm to find a point in a convex polytope.

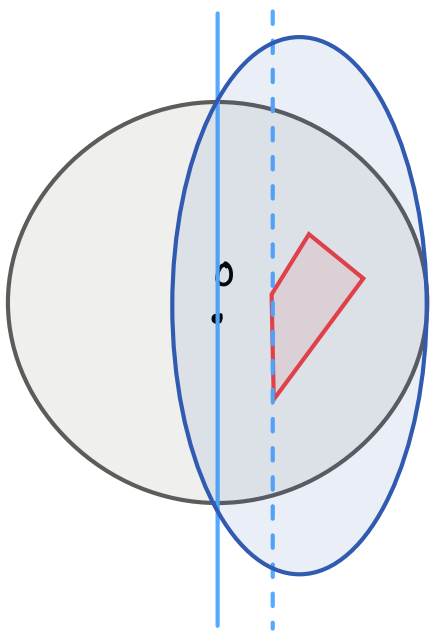
Ellipsoid Algorithm for finding points in Polytopes

The main idea here is to iteratively find ellipsoids where the density of the polytope is larger and larger until a point is found.

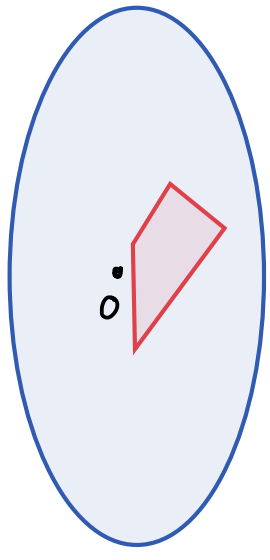


FACT 1 implies that the polytope is inside a ball of radius $R \leq 2^{poly(T)}$ centered at Origin.

- We first check if 0 is in the polytope



- If not, then we find some inequality that is violated, shift it to the origin and find an ellipsoid that contains the half-sphere containing the polytope



- Shift to the center of that ellipsoid
- Now volume of polytope relative to this ellipsoid is larger

We repeat the above for poly(T) iterations and if we haven't found a point in P by then, we say LP is not feasible.

The pseudocode is given as follows:

1. Let E be a ball of radius R containing polytope P
2. If $0 \in P$, output 0
3. Otherwise, find half-sphere containing P and new ellipsoid E' containing that half-sphere
4. $E \leftarrow E'$ and repeat the above for $M = \text{poly}(T)$ iterations
5. If we have not terminated, output "infeasible".

To prove that this works, we will prove the following lemma

LEMMA Let E' be the ellipsoid as above. Then,

$$\frac{\text{vol}(E')}{\text{vol}(E)} \leq e^{-\frac{1}{2(n+1)}} \approx 1 - \frac{1}{2(n+1)}$$

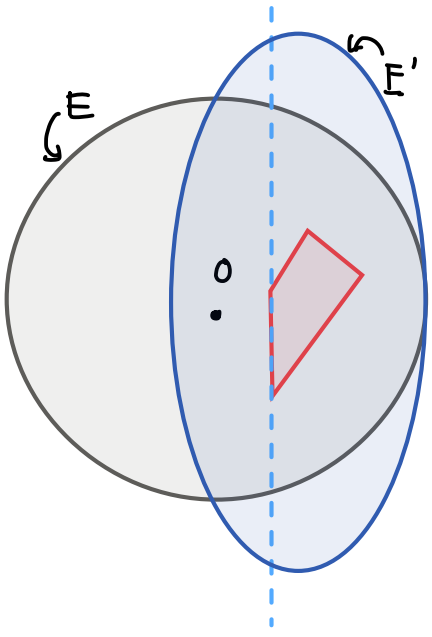
This implies that $\frac{\text{vol}(P)}{\text{vol}(E')} \geq e^{\frac{1}{2(n+1)}} \frac{\text{vol}(P)}{\text{vol}(E)}$

After t iterations, $\frac{\text{vol}(P)}{\text{vol}(E_{fin})} \geq e^{\frac{t}{2(n+1)}} \underbrace{\frac{\text{vol}(P)}{\text{vol}(E_{in})}}$

FACT 2 implies that if LP is feasible, this is at least $2^{-\text{poly}(T)}$

So, after $\text{poly}(T)$ iterations, we can be sure.

Proof of LEMMA Note that $E = \{ x \in \mathbb{R}^n \mid \sum_{i=1}^{n} x_i^2 \leq 1 \}$



E' = ellipsoid containing the right half ball
(otherwise we can just rotate)

We claim $E' = \left\{ x \;\middle|\; \left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2} \sum_{i=2}^{n} x_i^2 \leq 1 \right\}$

contains the half-ball

To see this, let $x \in E$, $x_1 \geq 0$, then

$$\left(\frac{n+1}{n}\right)^2 \left(x_1 - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2} \sum_{i=2}^{n} x_i^2$$

$$= \left(\frac{(n+1)x_1^2 - 1}{n}\right)^2 + \frac{n^2-1}{n^2} \sum_{i=2}^{n} x_i^2$$

$$= \frac{(n^2+2n+1)x_1^2 - 2(n+1)x_1 + 1}{n^2} + \frac{n^2-1}{n^2}\sum_{i=2}^{n} x_i^2$$

$$= \frac{(2n+2)x_1^2 - (2n+2)x_1}{n} + \frac{1}{n^2} + \frac{n^2-1}{n^2}\sum_{i=2}^{n} x_i^2$$

$$= \underbrace{\frac{(2n+2)x_1(x_1-1)}{n}}_{\leq 0} + \frac{1}{n^2} + \frac{n^2-1}{n^2}\underbrace{\sum_{i=2}^{n} x_i^2}_{\leq 1}$$

since $x_1 \in [0,1]$

$$\leq \frac{1}{n^2} + \frac{n^2-1}{n^2} \leq 1$$

Thus, $E \subseteq E'$.

Next, we claim that $\frac{\text{vol}(E')}{\text{vol}(E)} \leq e^{-\frac{1}{2(n+1)}}$

Recall $E = \{x \mid \Sigma x_i^2 \leq 1\}$

$$E' = \left\{x \mid \left(x_1 - \frac{1}{n+1}\right)^2\left(\frac{n+1}{n}\right)^2 + \frac{n^2-1}{n^2}\sum_{i=2}^{n} x_i^2 \leq 1\right\}$$

and $\frac{\text{volume of an ellipsoid}}{\text{volume of a ball}} = a_1 \cdots a_n$ where $a_1, \ldots a_n$ are the length of each half-axis where ellipsoid $= \left\{\Sigma \frac{x_i^2}{a_i^2} \leq 1\right\}$

So, we get $\frac{\text{vol}(E')}{\text{vol}(E)} = \frac{n}{n+1}\left(\sqrt{\frac{n^2}{n^2-1}}\right)^{n-1}$

$$= \left(1 - \frac{1}{n+1}\right)\left(1 + \frac{1}{n^2-1}\right)^{\left(\frac{n-1}{2}\right)}$$

$$\leq e^{-\frac{1}{n+1}} \cdot e^{\frac{(n-1)/2}{n^2-1}} \qquad \text{since } 1+x \leq e^x$$

$$= e^{-\frac{1}{(n+1)}} \cdot e^{\frac{1}{2(n+1)}}$$

$$= e^{-\frac{1}{2(n+1)}}$$

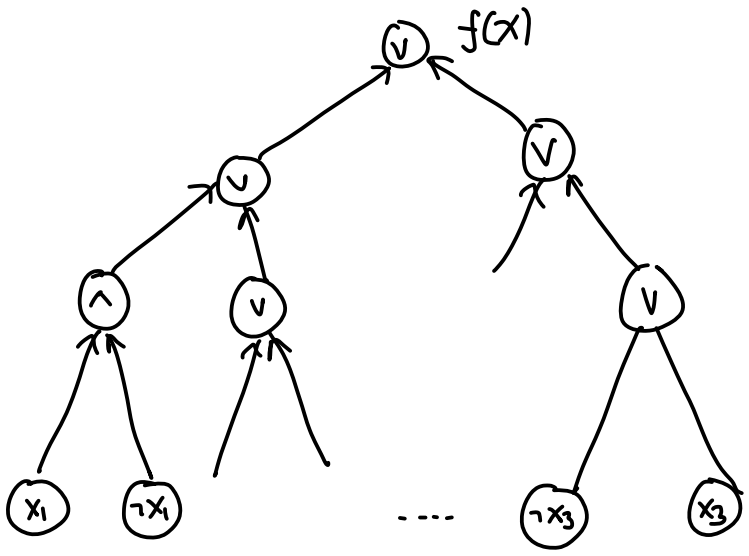
The one detail we skipped over is how to compute the ellipsoid in each step. One can explicitly write down the formulas in terms of the old ellipsoid and the inequality we have chosen and this involves matrix-vector multiplication but involves operations such as square-root. So, one has to be a bit careful in precision and numerical issues, so this algorithm is not quite practical.

However, ellipsoid algorithm is very powerful and can also solve linear programs that have exponentially many inequalities as long as we can find a violating constraint in polynomial time as well more general convex optimization problems.

Why is linear programming so powerful?

Linear programming is P-complete, which means that, any problem that has a polynomial time (deterministic) algorithm can be solved with linear programming.

To see this, let's recall boolean circuits — these are directed acyclic graphs where some of the nodes are labeled with x_i or $\neg x_i$ and the others are labeled with AND ($\wedge$) or OR ($\vee$).



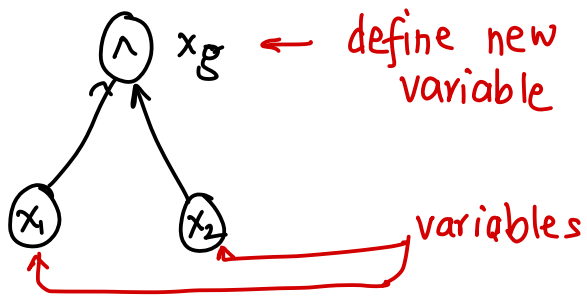
After topological sorting, the top node of the circuit computes some function of n bits, i.e.

$$f: \{0,1\}^n \longrightarrow \{0,1\}$$

Every efficient algorithm can be converted to a small circuit.

FACT If $f: \{0,1\}^n \to \{0,1\}$ can be computed in time T, then there is a circuit of size $O(T \log T)$ that computes it.

One can easily convert a circuit to a linear program as follows:



Add constraints:

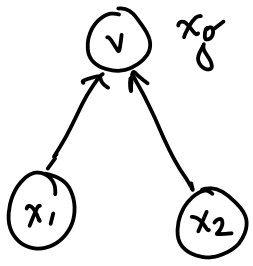
$$x_g \leq x_1$$
$$x_g \leq x_2$$
$$x_g \geq x_1 + x_2 - 1$$

$$\neg x_g = 1 - x_g$$

$$0 \leq x_g \leq 1$$

Objective

$\max x_v$ ← top node



$$x_g \geq x_1$$
$$x_g \geq x_2$$
$$x_g \leq x_1 + x_2$$

This linear program has polynomial size so we can compute f as well.