Suppose that we have the following LP with $n$ variables and $m$ constraints:

$$
\begin{align*}
\text{min} & \quad cx \\
\text{s.t.} & \quad Ax \leq b \\
& \quad x \geq 0
\end{align*}
$$

We want to find an algorithm that solves the LP in polynomial time (i.e. an algorithm that requires a number of computations bounded by a polynomial function of the input size). Note that each step takes into account the bit complexity of performing the operation.

1 Size of the Input

Assume input coefficients are all integer. Note that each integer $n$ is represented in binary with $1 + \lceil \log_2(|n|) + 1 \rceil$ bits. Then,

$$
\text{size}(n) = 1 + \lceil \log_2(|n|) + 1 \rceil
$$

Similarly,

$$
\text{size}(v) = \sum_{i=1}^{n} \text{size}(v_i) \quad \text{where } v \text{ is a vector and } v \in \mathbb{R}^{n \times 1}.
$$

$$
\text{size}(A) = \sum_{i,j} \text{size}(a_{ij}) \quad \text{where } A \text{ is a matrix.}
$$

For our LP, the total size is approximately

$$
L \sim \text{size}(A) + \text{size}(b) + \text{size}(c)
$$

2 Size of the Output

We now want to determine the size of the output. This is an important step, since if the output is exponential in the size of the input, it will take exponential time just to write it down.

To begin, note that we can bound the size of the input as $mnU$, where

$$
U = \max(\text{size}(d) : d \text{ is a component of } A, b, \text{or } c)
$$
Look at the size of the basic solution $x$. We know the solution to a system of equations of the form $\bar{A}x = \bar{b}$, where $\bar{A}$ is an $n \times n$ matrix, is given by Kramer’s rule

$$x_j = \frac{\det(\bar{A}_j)}{\det(\bar{A})}$$

where $\bar{A}_j$ is the matrix $\bar{A}$ with the $j^{th}$ column replaced by $\bar{b}$. In order to determine the size of the output, we want to bound the size of both determinants. In general, a determinant is defined as

$$\sum_{\sigma \in S_n} (-1)^{\text{sgn}(\sigma)} \frac{\leq U \text{ bits}}{\leq nU \text{ bits}} a_{1\sigma(2)}a_{1\sigma(2)}...a_{n\sigma(n)}$$

where the sum is computed over all permutations $\sigma$ of the numbers $\{1, 2, ..., n\}$. The formula contains $n!$ summands. Since adding two binary numbers with $m$ bits requires at most $m + 1$ bits to represent, adding $n!$ terms will require an additional $\log(n!)$ bits to represent. Using the inequality $\log(n!) \leq n\log(n)$ we get the following lemma:

**Lemma:** the number of bits in $\det(\bar{A})$ is less than $n\log(n) + nU$

### 3 Ellipsoid Algorithm for LP

We are given a polytope $P = \{x : Cx \leq d\}$. We want to find out whether there exists an $x \in P$. Suppose we have a black box that solves this problem and outputs a “yes” along with $x$ whenever there exists one (outputs “no” otherwise). How do we solve $\min\{cx : Ax \leq b, x \geq 0\}$ using this black box?

**Step 1:** Check if $\exists x$, such that $Ax \leq b, x \geq 0$. If the answer is no then the primal LP is infeasible. On the other hand if the answer is yes we go to step 2.

**Step 2:** Check if $\exists y$, such that $yA \geq c, y \geq 0$. If the answer is no then this dual is infeasible and since the primal LP is feasible, it is unbounded. If the answer is yes we go to step 3.

**Step 3:** Check if $Ax \leq b, x \geq 0$ and $yA \geq c, y \geq 0$ and $cx = by$. This outputs a solution $(x, y)$ such that $x$ is an optimal solution for the original LP.

Suppose that we have a method that only outputs “yes” or “no”, whereas in the previous setting when the answer was “yes”, we also obtain a feasible solution $x$. We can view this an merely trying to find a full rank set of tight constraints $P = \{x : Cx \leq d\}$ for a feasible vertex solution.

**Method:**

Denote $I$: The index set of tight constraints. (Initially this set $I = \emptyset$)

Now for $i = 1, \ldots, m$ consider the system of equations

- $C_jx \leq d_j$ for $j = i + 1, \ldots, m$
- $C_ix = d_i$
- $C_jx = d_j$ for $j \in I$
and check if this is feasible. If yes, add \( i \) to \( I \); if no, we delete the constraint \( i \). Now we continue until we have a system of equations that can be solved by standard Gaussian elimination.

Now we turn to an overview of the Ellipsoid method. The basic operation that we make use of is as follows: given a point \( a_k \), we can determine whether it satisfies all the constraints \( C_jx \leq d_j \forall j \).

If it does satisfy the constraints, then \( a_k \) is feasible. If it doesn’t, then we are able to identify some \( j \) such that \( C_jx > d_j \).

First of all, we assume that the feasible region \( P \) has non-zero volume. We start with an ellipsoid with some point \( a_k \) as its center such that the ellipsoid contains the whole of the feasible region \( P \). If \( a_k \) satisfies all the constraints, we are done. Otherwise, there exists some \( j \) such that \( C_ja_k > d_j \).

Now we divide the original ellipsoid into two half ellipsoids such that the feasible region remains entirely in one of the half ellipsoids. We do this by dividing the ellipsoid by a hyperplane that passes though \( a_k \) (the center of the ellipsoid) and is parallel to the constraint \( C_jx = d_j \), the constraint that is violated by \( a_k \). Now we compute a new ellipsoid that contains the good half-ellipsoid (and hence all feasible points, if any). We repeat the same procedure starting with the center of this new ellipsoid as a point to be checked for feasibility until the center of our new ellipsoid is within the feasible region.

Intuitively, if \( L \) is the size of our LP, and the initial volume of the ellipsoid is of the order \( 2^L \), then one iteration of our algorithm decreases the volume of our ellipsoid by roughly a factor of 2. So it seems that with \( 8L \) iterations we can obtain an ellipsoid with volume of \( 4^{-L} \) (which will be small enough to conclude that no feasible region exists at all).