

Complexity Issue and the Ellipsoid (Kachiyan) Method

Yinyu Ye

Department of Management Science and Engineering

Stanford University

Stanford, CA 94305, U.S.A.

<http://www.stanford.edu/~yyye>

How Good is the Simplex Method

Very good on *average*, but the *worse case* ...?

When the simplex method is used to solve a linear program the number of iterations to solve the problem starting from a basic feasible solution is typically a small multiple of m , e.g., between $2m$ and $3m$.

At one time researchers believed—and attempted to prove—that the simplex algorithm (or some variant thereof) always requires a number of iterations that is bounded by a *polynomial expression* in the problem size.

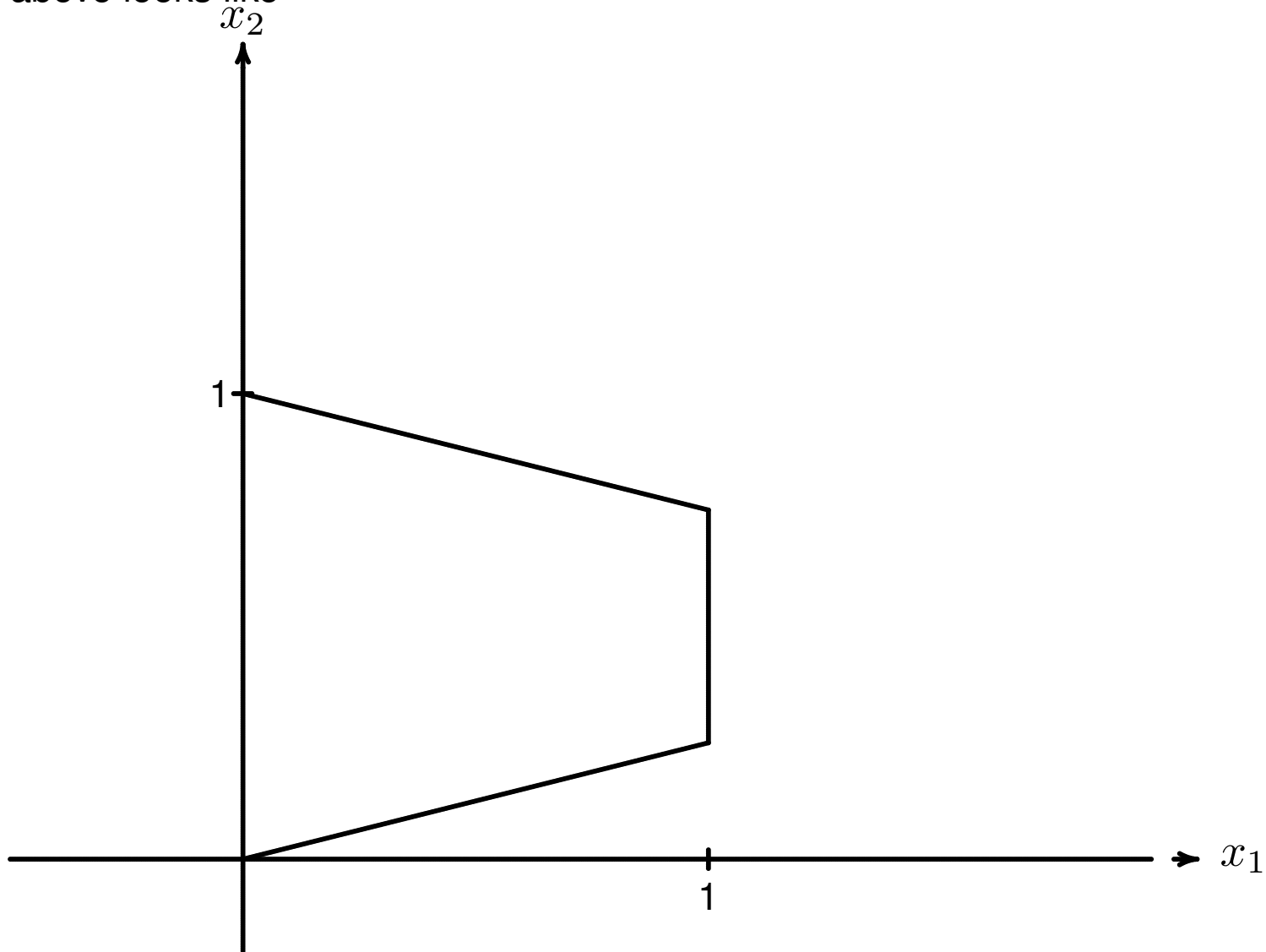
Klee and Minty Example

Consider

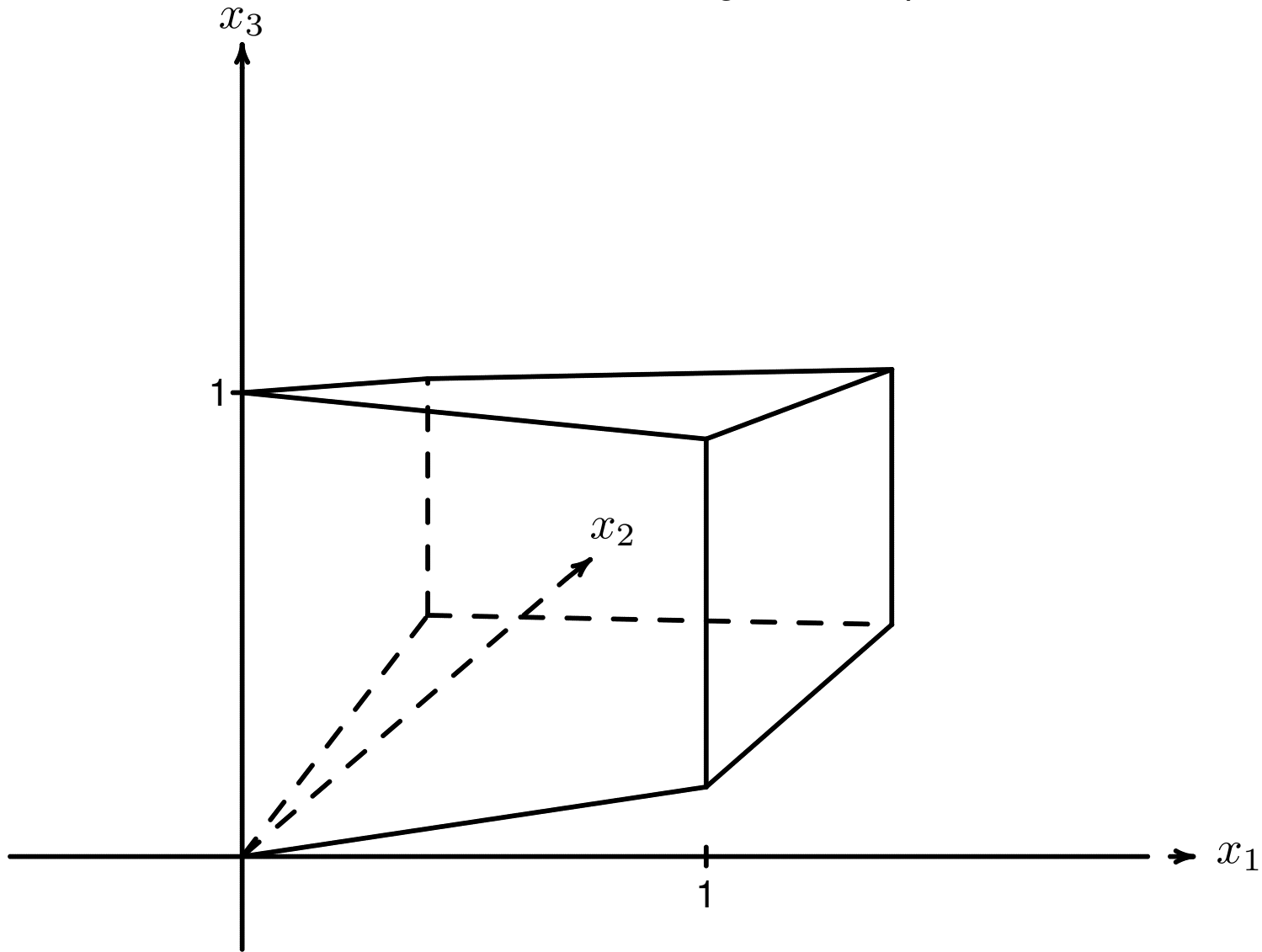
$$\begin{aligned} \max \quad & x_n \\ \text{subject to} \quad & x_1 \geq 0 \\ & x_1 \leq 1 \\ & x_j \geq \epsilon x_{j-1} \quad j = 2, \dots, n \\ & x_j \leq 1 - \epsilon x_{j-1} \quad j = 2, \dots, n \end{aligned}$$

where $0 < \epsilon < 1/2$. This presentation of the problem emphasizes the idea (see the figures below) that the feasible region of the problem is a **perturbation** of the **n -cube**.

In the case of $n = 2$ and $\epsilon = 1/4$, the feasible region of the linear program above looks like



For the case where $n = 3$, the feasible region of the problem above looks like



The formulation above does not immediately reveal the standard form representation of the problem. Instead, we consider a different one, namely

$$\begin{aligned}
 \max \quad & \sum_{j=1}^n 10^{n-j} x_j \\
 \text{subject to} \quad & 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1} \quad i = 1, \dots, n \\
 & x_j \geq 0 \quad j = 1, \dots, n
 \end{aligned}$$

The problem above^a also be used is easily cast as a linear program in standard form. Unfortunately, it is less apparent how to exhibit the relationship between its feasible region and a **perturbation** of the unit cube.

^aIt should be noted that there is no need to express this problem in terms of powers of 10. Using any constant $C > 1$ would yield the same effect (an **exponential number** of pivot steps).

Example

$$\begin{array}{llllll} \max & 100x_1 & + & 10x_2 & + & x_3 \\ \text{subject to} & x_1 & & & & \leq 1 \\ & 20x_1 & + & x_2 & & \leq 100 \\ & 200x_1 & + & 20x_2 & + & x_3 \leq 10,000 \end{array}$$

In this case, we have three constraints and three variables (along with their non-negativity constraints). After adding **slack variables**, we get a problem in standard form. The system has $m = 3$ equations and $n = 6$ nonnegative variables. In **tableau form**, the problem is

T^0

	$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
	1	100	10	1	0	0	0	0
	0	1	0	0	1	0	0	1
	0	20	1	0	0	1	0	100
	0	200	20	1	0	0	1	10,000
					•	•	•	

The bullets below the tableau indicate the columns that are basic.

T^1

	$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
1	0	10	1	-100	0	0	0	-100
0	1	0	0	1	0	0	0	1
0	0	1	0	-20	1	0	0	80
0	0	20	1	-200	0	1	1	9,800

●
●
●

T^2

	$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
1	0	0	1	100	-10	0	0	-900
0	1	0	0	1	0	0	0	1
0	0	1	0	-20	1	0	0	80
0	0	0	1	200	-20	1	1	8,200

●
●
●

T^3

	$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
1	-100	0	1	0	-10	0	0	-1,000
0	1	0	0	1	0	0	0	1
0	20	1	0	0	1	0	0	100
0	-200	0	1	0	-20	1	1	8,000

●
●
●

T^4

	$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
	1	100	0	0	0	10	-1	-9,000
	0	1	0	0	1	0	0	1
	0	20	1	0	0	1	0	100
	0	-200	0	1	0	-20	1	8,000
			●	●	●			

T^5

$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
1	0	0	0	-100	10	-1	-9,100
0	1	0	0	1	0	0	1
0	0	1	0	-20	1	0	80
0	0	0	1	200	-20	1	8,200

● ● ●

T^6

$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
1	0	-10	0	100	0	-1	-9,900
0	1	0	0	1	0	0	1
0	0	1	0	-20	1	0	80
0	0	20	1	-200	0	1	9,800

● ● ●

$$T^7$$

$-z$	x_1	x_2	x_3	x_4	x_5	x_6	1
1	-100	-10	0	0	0	-1	-10,000
0	1	0	0	1	0	0	1
0	20	1	0	0	1	0	100
0	200	20	1	0	0	1	10,000
			•	•	•		

$$(x_1, x_2, x_3, x_4, x_5, x_6) = (0, 0, 10^4, 1, 10^2, 0)$$

is **optimal** and that the objective function value is 10,000.

Along the way, we made $2^3 - 1 = 7$ **pivot steps**. The objective function made a **strict increase** with each change of basis.

Remark. The instance of the linear program (1) in which $n = 3$ leads to $2^3 - 1$ pivot steps when the **greedy rule** is used to select the pivot column. The general problem of the class (1) takes $2^n - 1$ pivot steps. To get an idea of how bad this can be, consider the case where $n = 50$. Now $2^{50} - 1 \approx 10^{15}$. In a year with 365 days, there are approximately 3×10^7 seconds. If a computer were running continuously and performing T iterations of the Simplex Algorithm per second, it would take approximately

$$\frac{10^{15}}{3T \times 10^8} = \frac{1}{3T} \times 10^8 \text{ years}$$

to solve the problem using the Simplex Algorithm with the greedy pivot selection rule.

An interesting connection

Consider the eight vectors $v^k = (v_1^k, v_2^k, v_3^k)$ where $k = 0, 1, \dots, 7$ and

$$v_j^k = \begin{cases} 1 & \text{if } x_j \text{ is basic in tableau } k \\ 0 & \text{otherwise} \end{cases}$$

Looking at the **eight tableaus** T^0, T^1, \dots, T^7 , we see that

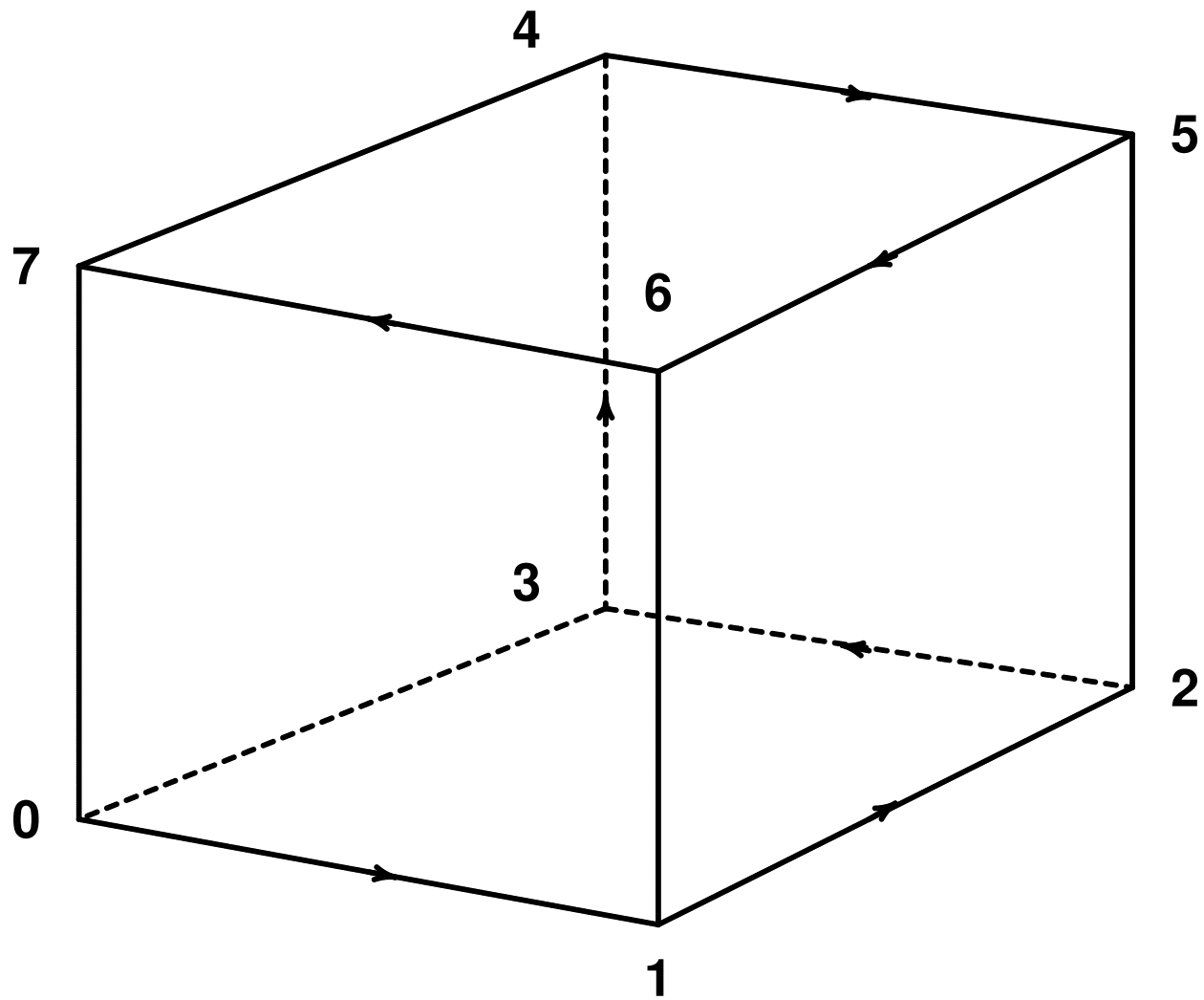
$$v^0 = (0, 0, 0) \quad v^4 = (0, 1, 1)$$

$$v^1 = (1, 0, 0) \quad v^5 = (1, 1, 1)$$

$$v^2 = (1, 1, 0) \quad v^6 = (1, 0, 1)$$

$$v^3 = (0, 1, 0) \quad v^7 = (0, 0, 1)$$

Now suppose we regard these vectors as the coordinates of the vertices of the 3-cube $[0, 1]$.



The figure above illustrates the fact that the **sequence of vectors** v^k corresponds to a path on the **edges** of the 3-cube. The path visits each **vertex** of the cube once and only once. Such a path is said to be **Hamiltonian**.

There is an amusing recreational literature that connects **Hamiltonian** path with certain **puzzles**. See Martin Gardner, “Mathematical games, the curious properties of the Gray code and how it can be used to solve puzzles,” *Scientific American* 227 (August 1972) pp. 106-109. See also, S.N. Afriat, *The Ring of Linked Rings*, London: Duckworth, 1982.

Elements of Complexity Theory

The term of **complexity** refers to the amount of resources required by a computation. Computational complexity wishes to associate to an algorithm more intrinsic measures of its time requirements

- a notion of **input size**,
- a set of **basic operations**, and
- a **cost** for each basic operations

The last two allow one to associate a (total) cost of a computation.

Polynomial Time Algorithms

bit size (**bit operations**) for integers, and **Unit size** (**unit cost**) for real numbers.

The former is usually referred to as the **Turing model of computation**, and latter is referred as the **real number arithmetic model**.

An algorithm is said to be a **polynomial-time** algorithm if its worst-case cost of computation is bounded above by a **polynomial** in the input size of the problem data.

Ellipsoid Method: the first polynomial-time algorithm for LP

The basic ideas of the **ellipsoid method** stem from research done in the nineteen sixties and seventies mainly in the Soviet Union (as it was then called) by others who preceded Khachiyan. The idea in a nutshell is to enclose the region of interest in each member of a sequence of ellipsoids whose size is decreasing, resembling the **bisection** method.

The significant contribution of Khachiyan was to demonstrate in two papers—published in 1979 and 1980—that under certain assumptions, the ellipsoid method constitutes a polynomially bounded algorithm for linear programming.

Linear Feasibility Problem

The method discussed here is really aimed at finding an element of a **polyhedral set** Y given by a system of linear inequalities.

$$Y = \{\mathbf{y} \in R^m : \mathbf{a}_j^T \mathbf{y} \leq c_j, \quad j = 1, \dots, n\}$$

Finding an element of Y can be thought of as being equivalent to solving a linear programming problem, though this requires a bit of discussion.

Two important assumptions

(A1) There is a vector $\mathbf{y}^0 \in \mathbb{R}^m$ and a scalar $R > 0$ such that the closed ball $S(\mathbf{y}^0, R)$ with center \mathbf{y}^0 and radius R , that is

$$\{\mathbf{y} \in \mathbb{R}^m : \|\mathbf{y} - \mathbf{y}^0\| \leq R\}$$

contains Y .

(A2) There is a known scalar $r > 0$ such that if Y is nonempty, then it contains a **ball** of the form $S(\mathbf{y}^*, r)$ with center at \mathbf{y}^* and radius r .

Note that this assumption implies that if Y is nonempty, then it has a nonempty **interior**.

Ellipsoid Representation

Ellipsoids are just sets of the form

$$E = \{\mathbf{y} \in \mathbb{R}^m : (\mathbf{y} - \bar{\mathbf{y}})^T B (\mathbf{y} - \bar{\mathbf{y}}) \leq 1\}$$

where $\bar{\mathbf{y}} \in \mathbb{R}^m$ is a given point (called the **center**) and B is a symmetric **positive definite** matrix of dimension m . We can use the notation $\text{ell}(\bar{\mathbf{y}}, B)$ to specify the ellipsoid E defined above.

Affine Transformation

Let us assume that $E_k = \text{ell}(\mathbf{y}^k, B_k^{-1})$, where the positive definite matrix B_k has the factorization $B_k = J_k J_k^T$. Now consider the transformation $\mathbf{y} \mapsto \mathbf{y}^k + J_k \mathbf{z}$.

Let $\mathbf{y} \in E_k$. Then $\mathbf{y} - \mathbf{y}^k = J_k \mathbf{z}$ for some vector $\mathbf{z} \in \mathbf{R}^m$. Now since $\mathbf{y} \in E_k$,

$$\begin{aligned}
 1 &\geq (\mathbf{y} - \mathbf{y}^k)^T B_k^{-1} (\mathbf{y} - \mathbf{y}^k) \\
 &= (J_k \mathbf{z})^T (\mathbf{J}_k \mathbf{J}_k^T)^{-1} (\mathbf{J}_k \mathbf{z}) \\
 &= \mathbf{z}^T \mathbf{J}_k^T (\mathbf{J}_k^T)^{-1} \mathbf{J}_k^{-1} \mathbf{J}_k \mathbf{z} \\
 &= \mathbf{z}^T \mathbf{z}
 \end{aligned}$$

so $\mathbf{z} \in \mathbf{S}(\mathbf{0}, \mathbf{1})$. Conversely, every such point maps to an element of E_k .

Cutting Plane

$$\text{vol}(E_k) = (\det B_k)^{1/2} \text{vol}(S(\mathbf{0}, 1)).$$

At each iteration of the algorithm, we will have $Y \subset E_k$. It is then possible to check whether $\mathbf{y}^k \in Y$. If so, we have found an element of Y as required. If not, there is at least one constraint that is violated. Suppose $\mathbf{a}_j^T \mathbf{y}^k > c_j$. Then

$$Y \subset \frac{1}{2}E_k := \{\mathbf{y} \in E_k : \mathbf{a}_j^T \mathbf{y} \leq \mathbf{a}_j^T \mathbf{y}^k\}$$

This set is “half the ellipsoid” cut through its center.

New Containing Ellipsoid

The **successor ellipsoid** E_{k+1} is constructed as follows. Define

$$\tau = \frac{1}{m+1}, \quad \delta = \frac{m^2}{m^2-1}, \quad \sigma = 2\tau.$$

Let

$$\mathbf{y}^{k+1} = \mathbf{y}^k - \frac{\tau}{(\mathbf{a}_j^\top B_k \mathbf{a}_j)^{1/2}} B_k \mathbf{a}_j, \quad B_{k+1} = \delta \left(B_k - \sigma \frac{B_k \mathbf{a}_j \mathbf{a}_j^\top B_k}{\mathbf{a}_j^\top B_k \mathbf{a}_j} \right).$$

Theorem 1 *The ellipsoid $E_{k+1} = \text{ell}(\mathbf{y}^{k+1}, B_{k+1}^{-1})$ defined as above is the ellipsoid of **least volume** containing $\frac{1}{2}E_k$. Moreover,*

$$\frac{\text{vol}E_{k+1}}{\text{vol}E_k} = \left(\frac{m^2}{m^2-1} \right)^{(m-1)/2} \frac{m}{m+1} < \exp \left(-\frac{1}{2(m+1)} \right) < 1.$$

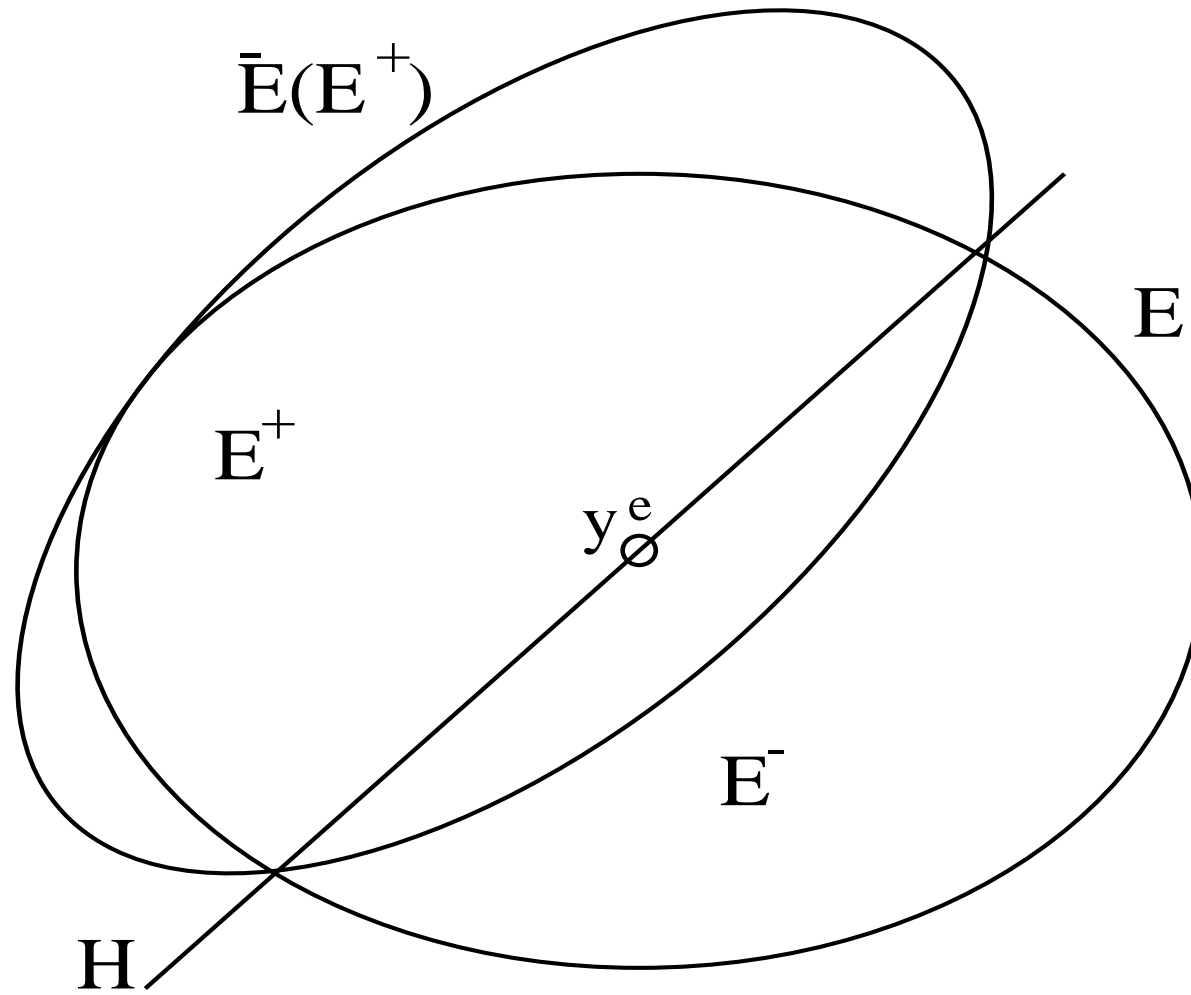


Figure 1: The least volume ellipsoid containing a half ellipsoid

The Ellipsoid Algorithm

Input: $A \in R^{m \times n}$, $\mathbf{c} \in R^n$, $\mathbf{y}^0 \in R^m$ such that Y (as defined on Slides 3-4) satisfies (A1) and (A2).

Output: $\mathbf{y} \in Y$.

Initialization: Set $B_0 = \frac{1}{R^2} I$, $K = \lceil 2m(m+1) \log(R/r) \rceil + 1$.

For $k = 0, 1, \dots, K - 1$ do

Iteration k : If $\mathbf{y}^k \in Y$, STOP: result is $\mathbf{y} = \mathbf{y}^k$. Otherwise, choose j with $\mathbf{a}_j^T \mathbf{y}^k > c_j$. Update \mathbf{y}^k and B_k (as defined on Slide 8).

Performance of the Ellipsoid Method

Under the assumptions stated above, the ellipsoid method solves linear programs in a polynomially bounded number of iterations. It is easy to see that the bound is $O(m^4 \log(R/r))$.

Computational experience shows that the number of iterations required to solve a linear programming problem is very close to the **theoretical upper bound**. This means that the method is **inefficient** in a practical sense. In contrast to this, although the simplex method is known to exhibit **exponential behavior** on specially constructed problems such as those of Klee and Minty, it normally requires a number of iterations that is a small multiple of the number of linear equations in the standard form of the problem.

Linear Programming

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^T \mathbf{x} \\ \text{(P)} \quad \text{subject to} & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array}$$

$$\begin{array}{ll} \text{minimize} & \mathbf{b}^T \mathbf{y} \\ \text{(D)} \quad \text{subject to} & A^T \mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{array}$$

$$\mathbf{c}^T \mathbf{x} \leq \mathbf{b}^T \mathbf{y}.$$

Integer Data

Next, we assume that the **data** for the problem are all **integers**. As a measure of the size of the problem above we let $c_j = a_{0j}$ and define

$$L = \sum_{i=0}^m \sum_{j=1}^n [\log_2(|a_{ij}| + 1) + 1].$$

In our discussion above, we made two assumptions about Y . One of the assumptions, (A2), effectively says that if Y is nonempty, then it possesses a nonempty interior. The **linear inequalities** are relaxed to

$$\mathbf{a}_j^T \mathbf{y} < c_j + 2^{-L} \quad j = 1, \dots, n. \quad (1)$$

It was shown by Gács and Lovasz (1981) that if the inequality system (1) has a solution, then so does

$$\mathbf{a}_j^T \mathbf{y} \leq c_j, \quad j = 1, \dots, n.$$

Bounds from L

Therefore, we can bound

$$r \geq 2^{-L}.$$

On the other hand, we can bound

$$R \leq O(2^L).$$

Thus,

$$\log(R/r) \leq O(L),$$

which is linear (polynomial) in L .

The sliding objective hyperplane method

Consider

$$\begin{aligned} & \text{minimize} && \mathbf{b}^T \mathbf{y} \\ \text{(D)} & \text{subject to} && A^T \mathbf{y} \geq \mathbf{c} \\ & && \mathbf{y} \geq \mathbf{0} \end{aligned}$$

At the center, \mathbf{y}^k , of the ellipsoid, if a constraint is violated then add the corresponding **constraint hyperplane** as the cut; otherwise, add **objective hyperplane** $\mathbf{b}^T \mathbf{y} \geq \mathbf{b}^T \mathbf{y}^k$ as the cut.

Desired Theoretical Properties

- **Separation Problem:** Either decide $\mathbf{x} \in P$ or find a vector \mathbf{d} such that $\mathbf{d}^T \mathbf{x} \leq \mathbf{d}^T \mathbf{y}$ for all $\mathbf{y} \in P$.
- **Oracle** to generate \mathbf{d} without **enumerating** all hyperplanes.

Theorem 2 *If the **separating (oracle)** problem can be solved in polynomial time of m and $\log(R/r)$, then we can solve the standard linear programming problem whose running time is polynomial in m and $\log(R/r)$ that is independent of n , the number of inequality constraints.*

LP with an exponentially large number of inequalities: TSP

Given an undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ where \mathcal{N} is the set of n nodes and length c_e for every edge $e \in \mathcal{E}$, the goal is find a tour (a cycle that visits all nodes) of minimal length. To model the problem, we define for every edge e a variable x_e , which is 1 if e is in the tour and 0 otherwise.

Let $\delta(i)$ be the set of edges incident to node i , then

$$\sum_{e \in \delta(i)} x_e = 2, \forall i \in \mathcal{N}.$$

Let $S \subset \mathcal{N}$ and

$$\delta(S) = \{e : e = (i, j), i \in S, j \notin S\}.$$

Then,

$$\sum_{e \in \delta(S)} x_e \geq 2, \forall S \subset \mathcal{N}, S \neq \emptyset, \mathcal{N}.$$

LP relaxation of TSP

$$\begin{array}{ll} \text{minimize} & \sum_{e \in \mathcal{E}} c_e x_e \\ \text{subject to} & \sum_{e \in \delta(i)} x_e = 2, \forall i \in \mathcal{N}, \\ & \sum_{e \in \delta(S)} x_e \geq 2, \forall S \subset \mathcal{N}, S \neq \emptyset, \mathcal{N}, \\ & 0 \leq x_e \leq 1 \forall e \in \mathcal{E}. \end{array}$$

(TSP)

This problem has an **exponential number** of inequalities since there are $2^n - 2$ of proper subsets of S

A oracle to check the separation

Given x_e^* we like to check if

$$\sum_{e \in \delta(S)} x_e^* \geq 2, \forall S \subset \mathcal{N}, S \neq \emptyset, \mathcal{N}.$$

Assign x_e^* as the capacity for every edge $e \in \mathcal{E}$, then the problem is to check if the **min-cut** of the graph is greater than or equal to **2**.

This problem can be formulated as **Maximum Flow** problems (how?) and can be solved as a small linear program.