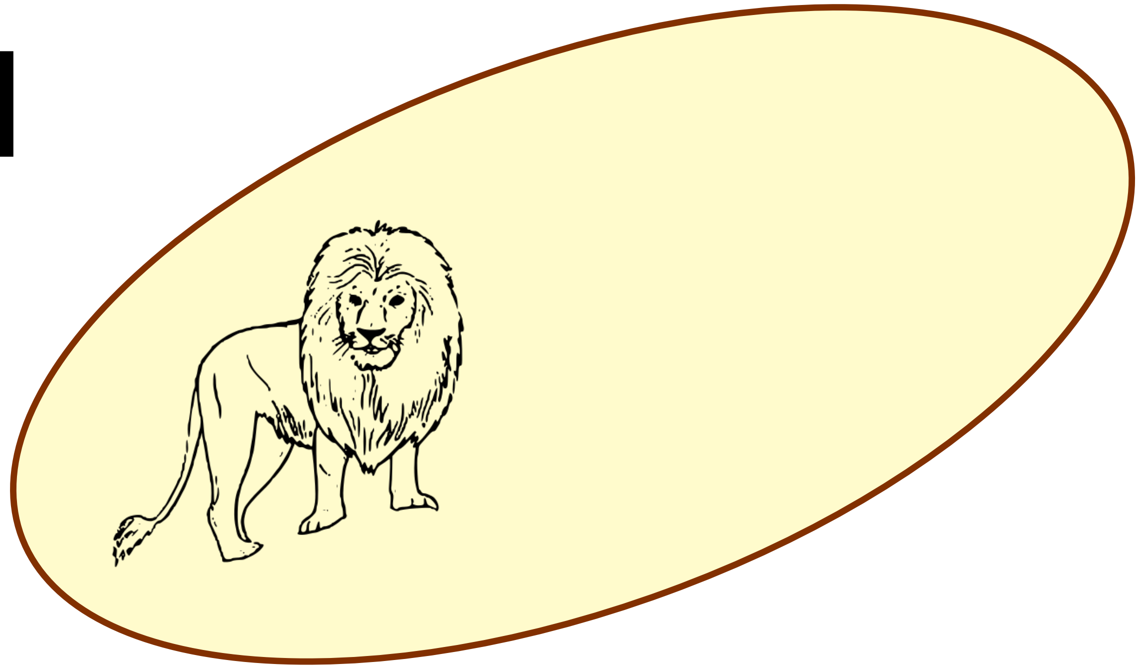
Other algorithms besides the simplex method



Other than Simplex?!

Dual simplex method

- Perform simplex method on the dual.
- Implementation details can lead to crucial speed-ups.
- Particularly suitable when $n - m \ll m$.

Other than Simplex?!

Dual simplex method

- Perform simplex method on the dual.
- Implementation details can lead to crucial speed-ups.
- Particularly suitable when $n - m \ll m$.

Primal-dual method

- through feasible solutions of dual, but via auxilliary problem instead of pivot.
- important for approximations of combinatorial optimizations.

Other than Simplex?!

Dual simplex method

- Perform simplex method on the dual.
- Implementation details can lead to crucial speed-ups.
- Particularly suitable when $n - m \ll m$.

Primal-dual method

- through feasible solutions of dual, but via auxilliary problem instead of pivot.
- important for approximations of combinatorial optimizations.

Ellipsoid method

- first polynomial-time algorithm for LP
- slow in practice

Other than Simplex?!

Dual simplex method

- Perform simplex method on the dual.
- Implementation details can lead to crucial speed-ups.
- Particularly suitable when $n - m \ll m$.

Primal-dual method

- through feasible solutions of dual, but via auxilliary problem instead of pivot.
- important for approximations of combinatorial optimizations.

Ellipsoid method

- first polynomial-time algorithm for LP
- slow in practice

Interior point methods

- class of methods, some polynomial-time
- Successfully competes with the simplex algorithm on large LPs

Other than Simplex?!

Dual simplex method

- Perform simplex method on the dual.
- Implementation details can lead to crucial speed-ups.
- Particularly suitable when $n - m \ll m$.

Primal-dual method

- through feasible solutions of dual, but via auxilliary problem instead of pivot.
- important for approximations of combinatorial optimizations.

Ellipsoid method

- first polynomial-time algorithm for LP
- slow in practice

today: sketch of polynomial-time algorithms

Interior point methods

- class of methods, some polynomial-time
- Successfully competes with the simplex algorithm on large LPs

Ellipsoid method

- History: Invented 1970 by Shor, Judin, Nemirovski for nonlinear problems

  1979: Lenoid Khachyian showed the ellipsoid method solves linear programs in polynomial time.

The New York Times

Similar Transportation Bond Issues Are Passed in New York and Jersey

CARTER SAID TO ASK FOR LOYALTY CHECK OF AIDES ON POLITICS

IRAN'S CIVIL GOVERNMENT OUT; HOSTAGES FACE DEATH THREAT; OIL EXPORTS BELIEVED HALTED

F.A.A. Seeks Fine of $1.5 Million For 'Unairworthy' Braniff Flights

Main Iran Oil Port Reported Closed And Prices of Spot Petroleum Soar

Energy Dept. Accuses 9 Refiners Of $1.18 Billion Oil Overcharges

A Soviet Discovery Rocks World of Mathematics

A Soviet Discovery Rocks World of Mathematics

From the archive, 29 October 1979: Russian way with the mathematical travelling salesman

The Guardian

Unknown Soviet mathematician LG Khachian offers a solution to a problem that has the computing world baffled

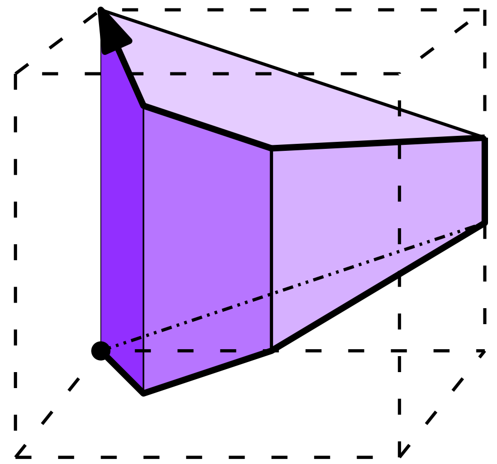
A 1970s mainframe computer: LG Khachian offered a solution to the 'travelling salesman' problem of computer processing. Photograph: ClassicStock/Alamy

Ellipsoid method

- History: Invented 1970 by Shor, Judin, Nemirovski for nonlinear problems

  1979: Lenoid Khachyian showed the ellipsoid method solves linear programs in polynomial time.

- Polynomial time! Theoretically important.
  - → Simplex method is not known to be polynomial
  - → most pivot rules are known not to be polynomial

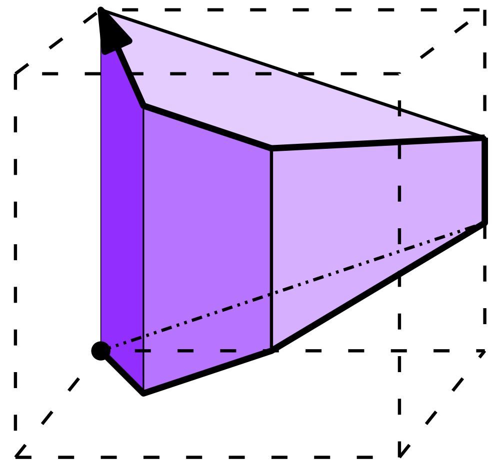


What is this?

Ellipsoid method

- History: Invented 1970 by Shor, Judin, Nemirovski for nonlinear problems

  1979: Lenoid Khachyian showed the ellipsoid method solves linear programs in polynomial time.



Klee-Minty cube

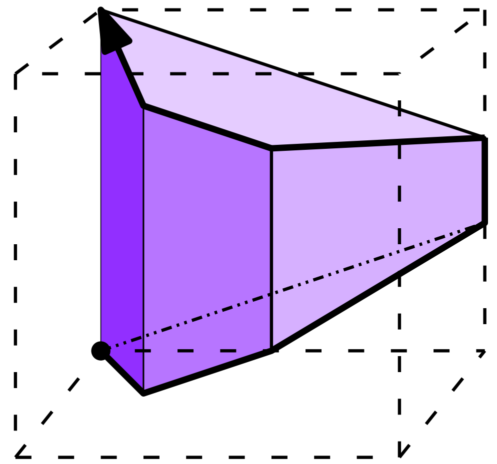
- Polynomial time! Theoretically important.
  - → Simplex method is not known to be polynomial
  - → most pivot rules are known not to be polynomial

Ellipsoid method

- History: Invented 1970 by Shor, Judin, Nemirovski for nonlinear problems

  1979: Lenoid Khachyian showed the ellipsoid method solves linear programs in polynomial time.

- Polynomial time! Theoretically important.
  - → Simplex method is not known to be polynomial
  - → most pivot rules are known not to be polynomial



Klee-Minty cube

- Not competitive in practice.
  - → Polynomial of high degree.

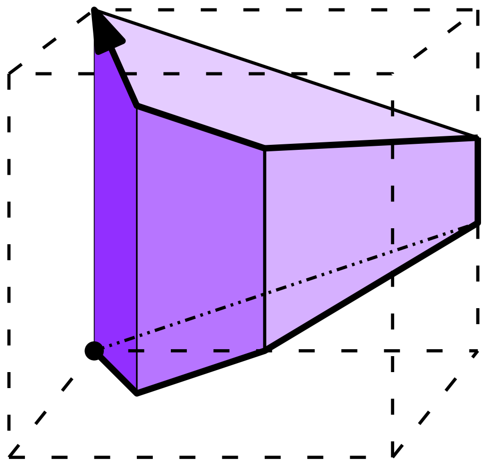
	Most problems	Worst case
Simplex method		
Ellipsoid method		

Ellipsoid method

- History: Invented 1970 by Shor, Judin, Nemirovski for nonlinear problems

  1979: Lenoid Khachyian showed the ellipsoid method solves linear programs in polynomial time.

- Polynomial time! Theoretically important.
  - → Simplex method is not known to be polynomial
  - → most pivot rules are known not to be polynomial



Klee-Minty cube

- Not competitive in practice.
  - → Polynomial of high degree.

	Most problems	Worst case
Simplex method	Extremely fast	Exponentially bad
Ellipsoid method	High degree polynomial	High degree polynomial

Polynomial and strongly polynomial algorithms

Polynomial running time in terms of the input size — but what is the input size?

Polynomial and strongly polynomial algorithms

Polynomial running time in terms of the input size — but what is the input size?

Notation

The bit size of an integer i is

$$\langle i \rangle = \lceil \log_2 (|i| + 1) \rceil + 1$$

This is the $\#$ of bits of i in binary, plus a sign.

Polynomial and strongly polynomial algorithms

Polynomial running time in terms of the input size — but what is the input size?

Notation

The bit size of an integer i is

$$\langle i \rangle = \lceil \log_2 (|i| + 1) \rceil + 1$$

This is the $\#$ of bits of i in binary, plus a sign.

Example: $\langle 1234 \rangle =$???

Polynomial and strongly polynomial algorithms

Polynomial running time in terms of the input size — but what is the input size?

Notation

The bit size of an integer i is

$$\langle i \rangle = \lceil \log_2 (|i| + 1) \rceil + 1$$

This is the $\#$ of bits of i in binary, plus a sign.

Example: $\langle 1234 \rangle = 11 + 1 = 12$ since 1234 is 10011010010 in binary.

Polynomial and strongly polynomial algorithms

Polynomial running time in terms of the input size — but what is the input size?

Notation

The bit size of an integer i is

$$\langle i \rangle = \lceil \log_2 (|i| + 1) \rceil + 1$$

This is the $\#$ of bits of i in binary, plus a sign.

Example: $\langle 1234 \rangle = 11 + 1 = 12$ since 1234 is 10011010010 in binary.

For $r = p/q$ rational, we define $\langle r \rangle = \langle p \rangle + \langle q \rangle$.

→ Similar for vectors and matrices.

Polynomial and strongly polynomial algorithms

A linear program Maximize $c^T x$ subject to $Ax \leq b$ with rational entries has size $\langle A \rangle + \langle b \rangle + \langle c \rangle$.

Example: $A = \begin{bmatrix} 3 & -1 & 2 \\ 5 & 7 & 4 \end{bmatrix}$ $b = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$ $c = \begin{bmatrix} 1,000,521 \\ 3,012,554 \\ 19,728,213 \end{bmatrix}$

Polynomial and strongly polynomial algorithms

A linear program Maximize $c^T x$ subject to $Ax \leq b$ with rational entries has size $\langle A \rangle + \langle b \rangle + \langle c \rangle$.

Example: $A = \begin{bmatrix} 3 & -1 & 2 \\ 5 & 7 & 4 \end{bmatrix}$ $b = \begin{bmatrix} 2 \\ 5 \end{bmatrix}$ $c = \begin{bmatrix} 1,000,521 \\ 3,012,554 \\ 19,728,213 \end{bmatrix}$

Def.: An algorithm for linear programming is polynomial if there is a polynomial $p(\chi)$ such that for all linear programs with rational coefficients, the algorithm requires at most $p(\langle A \rangle + \langle b \rangle + \langle c \rangle)$ steps.

Polynomial and strongly polynomial algorithms

A linear program Maximize $c^T x$ subject to $Ax \leq b$ with rational entries has size $\langle A \rangle + \langle b \rangle + \langle c \rangle$.

Example: $$A = \begin{bmatrix} 3 & -1 & 2 \\ 5 & 7 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 2 \\ 5 \end{bmatrix} \qquad c = \begin{bmatrix} 1,000,521 \\ 3,012,554 \\ 19,728,213 \end{bmatrix}$$

Def.: An algorithm for linear programming is polynomial if there is a polynomial $p(\chi)$ such that for all linear programs with rational coefficients, the algorithm requires at most $p(\langle A \rangle + \langle b \rangle + \langle c \rangle)$ steps.

Steps are in terms of bit operations — adding two k bit integers requires at least k steps!

Digression: Gaussian Elimination

Solve $Ax = b$, A size $n \times n$, rational entries.

→ At most Cn^3 arithmetic operations.

$$\begin{bmatrix} 2 & 0 & 0 & \cdots & 0 \\ 1 & 2 & 0 & \cdots & 0 \\ 1 & 1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 & 2 \end{bmatrix}$$

Digression: Gaussian Elimination

Solve $Ax = b$, A size $n \times n$, rational entries.

→ At most Cn^3 arithmetic operations.

$$\begin{bmatrix} 2 & 0 & 0 & \cdots & 0 \\ 1 & 2 & 0 & \cdots & 0 \\ 1 & 1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 & 2 \end{bmatrix}$$

How long does naive Gaussian elimination take on this matrix if you only use integers (no fractions)?

Digression: Gaussian Elimination

Solve $Ax = b$, A size $n \times n$, rational entries.

→ At most Cn^3 arithmetic operations.

$$\begin{bmatrix} 2 & 0 & 0 & \cdots & 0 \\ 1 & 2 & 0 & \cdots & 0 \\ 1 & 1 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 & 2 \end{bmatrix}$$

How long does naive Gaussian elimination take on this matrix if you only use integers (no fractions)?

Naive implementations can produce entries requiring 2^n bits, giving an exponential algorithm!

But smarter implementations are indeed polynomial.

Ellipsoid and interior point methods are polynomial, simplex method is not.

Strongly polynomial algorithms

Strongly polynomial algorithms are polynomial in the sense defined before,
and also are polynomial in the $\#$ of arithmetic operations.

Strongly polynomial algorithms

Strongly polynomial algorithms are polynomial in the sense defined before,
and also are polynomial in the $\#$ of arithmetic operations.

Gaussian elimination (non-naive) is strongly polynomial:
number of arithmetic operations is bounded by Cn^3, independent of size of numbers

Strongly polynomial algorithms

Strongly polynomial algorithms are polynomial in the sense defined before,
and also are polynomial in the $\#$ of arithmetic operations.

Gaussian elimination (non-naive) is strongly polynomial:
  number of arithmetic operations is bounded by Cn^3, independent of size of numbers

A strongly polynomial algorithm for linear programming would be polynomial (as defined before) and require at most $p[n+m]$ arithmetic operations for some polynomial p.

Strongly polynomial algorithms

Strongly polynomial algorithms are polynomial in the sense defined before,
and also are polynomial in the $\#$ of arithmetic operations.

Gaussian elimination (non-naive) is strongly polynomial:
number of arithmetic operations is bounded by Cn^3, independent of size of numbers

A strongly polynomial algorithm for linear programming would be polynomial (as defined before) and require at most $p[n+m]$ arithmetic operations for some polynomial p.

No strongly polynomial algorithm for linear programming is known!

Strongly polynomial algorithms

Strongly polynomial algorithms are polynomial in the sense defined before,
and also are polynomial in the $\#$ of arithmetic operations.

Gaussian elimination (non-naive) is strongly polynomial:
number of arithmetic operations is bounded by Cn^3, independent of size of numbers

A strongly polynomial algorithm for linear programming would be polynomial (as defined before) and require at most $p[n+m]$ arithmetic operations for some polynomial p.

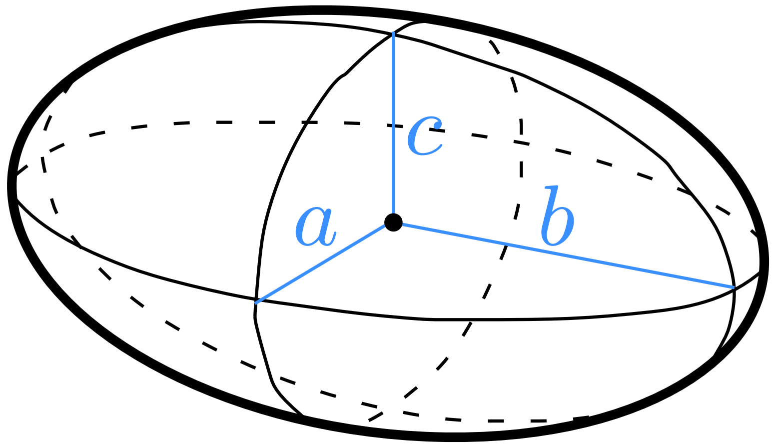
No strongly polynomial algorithm for linear programming is known!

Ellipsoid method is not: For all $K \in \mathbb{N}$, we can find a LP with 2 variables, 2 constraints, and ellipsoid method requires $\geq K$ arithmetic steps!

The bit sizes in these examples go to ∞ with K: $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ $\langle A \rangle \to \infty$ as $K \to \infty$

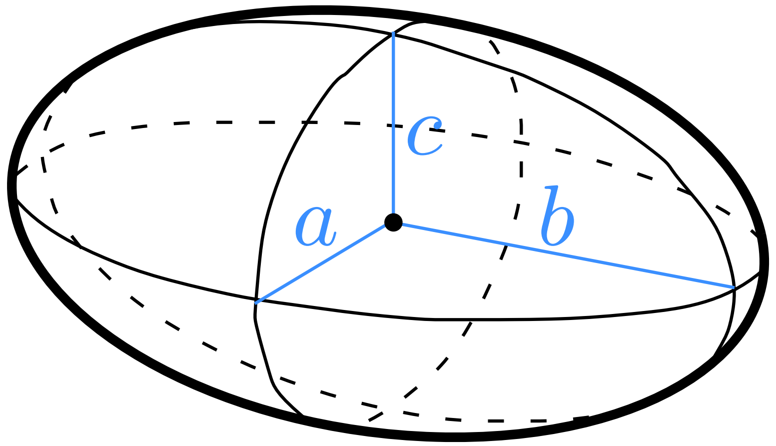
The Ellipsoid Method

- First polynomial time algorithm for linear programming
- Slow in practice



The Ellipsoid Method

- First polynomial time algorithm for linear programming
- Slow in practice
- We will show how to find a feasible solution to $Ax \leq b$



Recall:

In some sense finding an optimal solution is no harder than finding a feasible solution.

Finding an optimal solution to

Maximize $c^T x$ subject to $Ax \leq b$, $x \geq 0$

is the same as finding a feasible solution to

~~Maximize $c^T x$~~

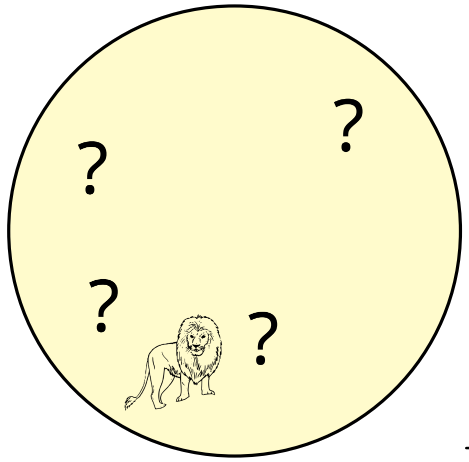
subject to

$$
\begin{aligned}
Ax &\leq b \\
A^T y &\geq c \\
c^T x &\geq b^T y \\
x &\geq 0, y \geq 0
\end{aligned}
$$

The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

Find lion in Sahara

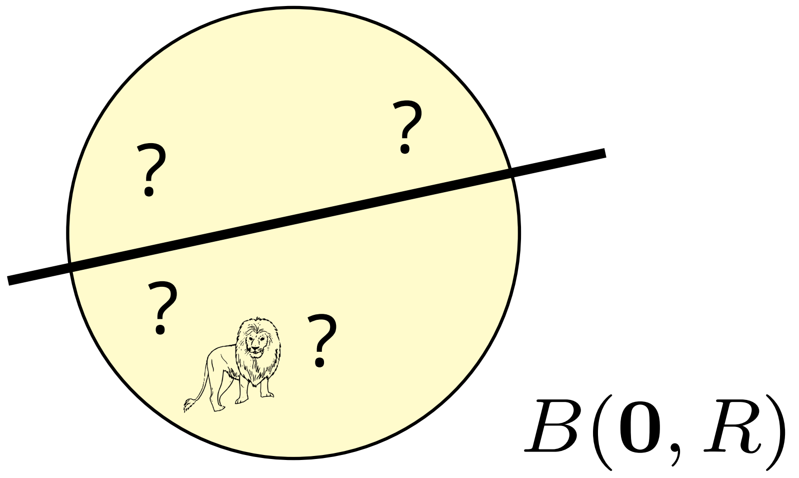


$B(\mathbf{0}, R)$

The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

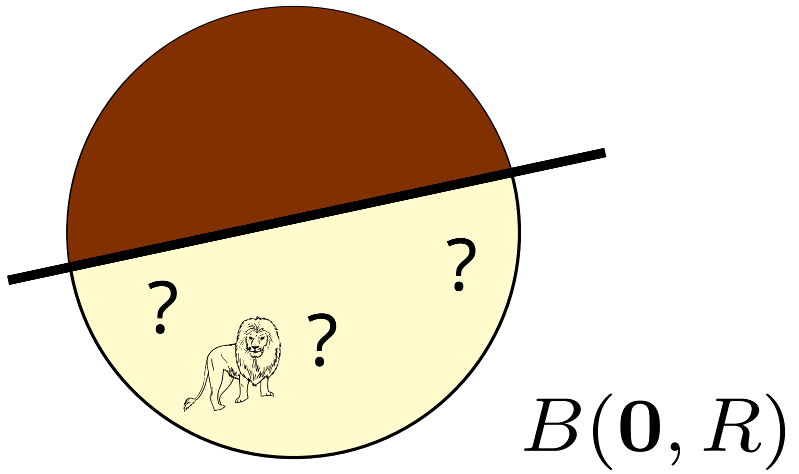
Find lion in Sahara



The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

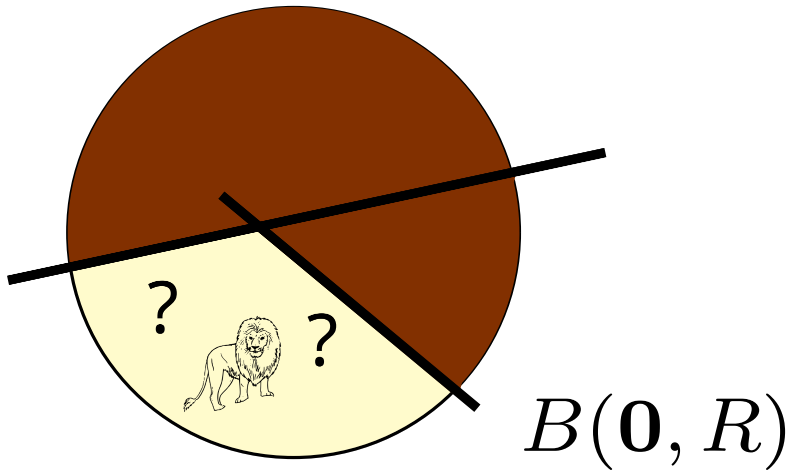
Find lion in Sahara



The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

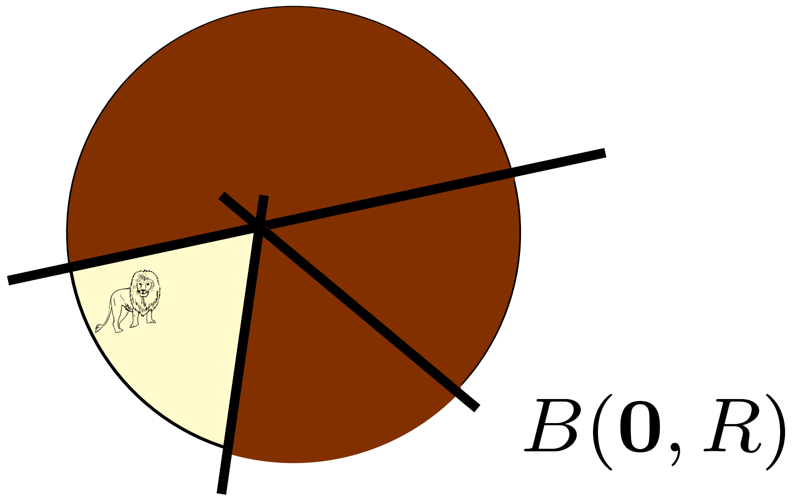
Find lion in Sahara



The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

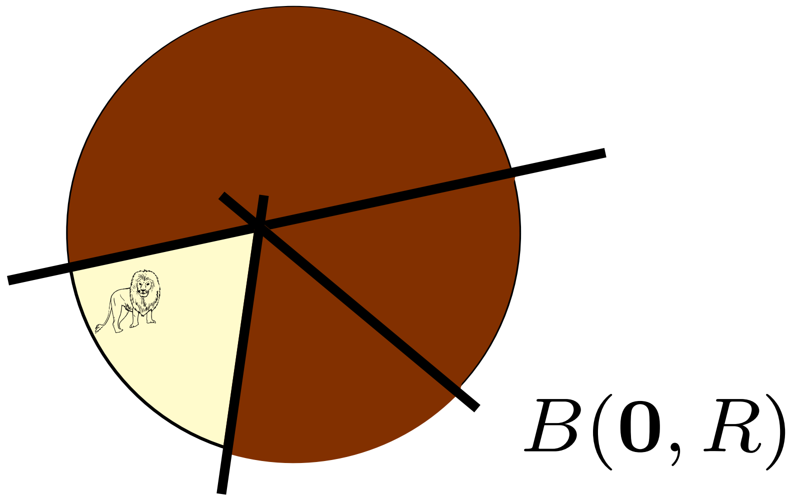
Find lion in Sahara



The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

Find lion in Sahara

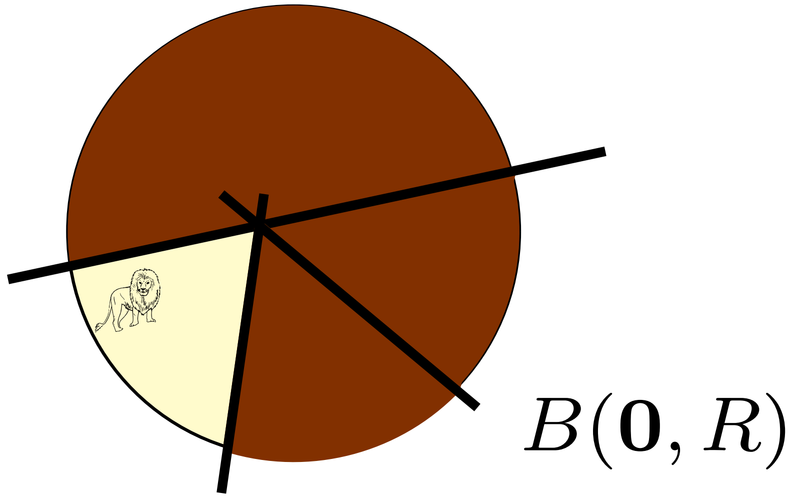


Cannot find arbitrary small lions

The Ellipsoid Method

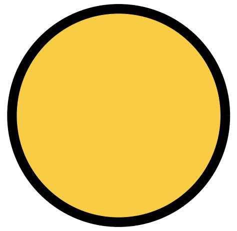
Analogy: Find feasible solution to $Ax \leq b$

Find lion in Sahara



Cannot find arbitrary small lions

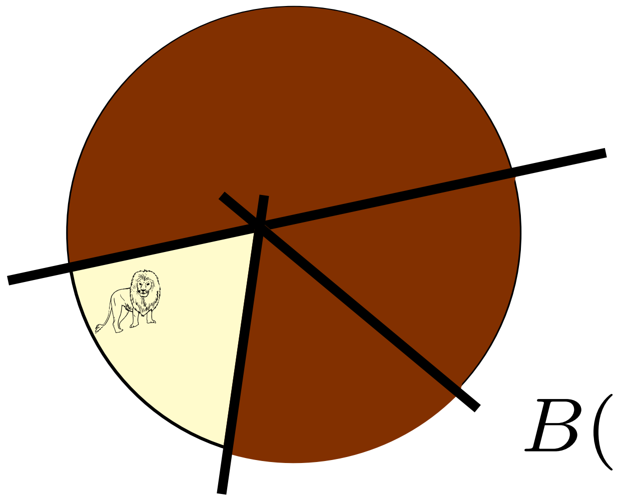
Looking for a lion of size at least an ε-ball



The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

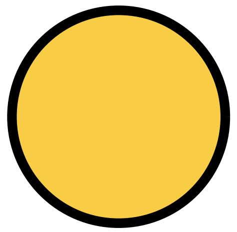
Find lion in Sahara



$B(\mathbf{0}, R)$

Cannot find arbitrary small lions

Looking for a lion of size at least an ε-ball



Let $\varphi = \langle A \rangle + \langle b \rangle$ be the input size of $Ax \leq b$.

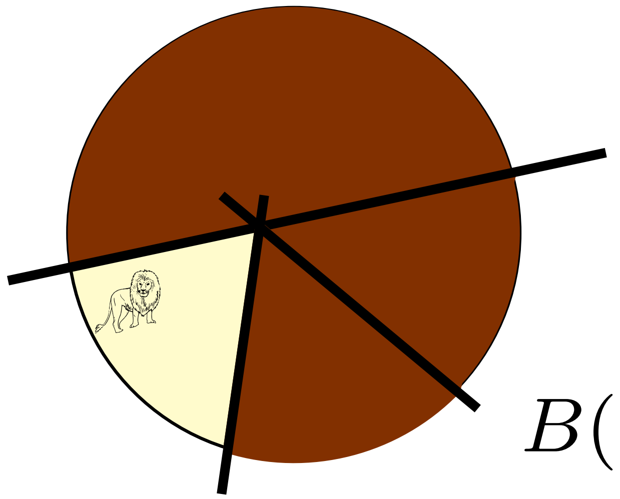
Let $\eta = 2^{-5\varphi}, \varepsilon = 2^{-6\varphi}$. Then:

$Ax \leq b$ has a solution $\iff$ $Ax \leq b + \eta$ has a solution (indeed a full ε-ball).

The Ellipsoid Method

Analogy: Find feasible solution to $Ax \leq b$

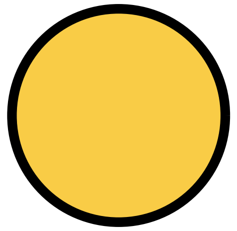
Find lion in Sahara



$B(\mathbf{0}, R)$

Cannot find arbitrary small lions

Looking for a lion of size at least an ε-ball



Let $\varphi = \langle A \rangle + \langle b \rangle$ be the input size of $Ax \leq b$.

Let $\eta = 2^{-5\varphi}, \varepsilon = 2^{-6\varphi}$. Then:

$Ax \leq b$ has a solution $\iff$ $Ax \leq b + \eta$ has a solution (indeed a full ε-ball).

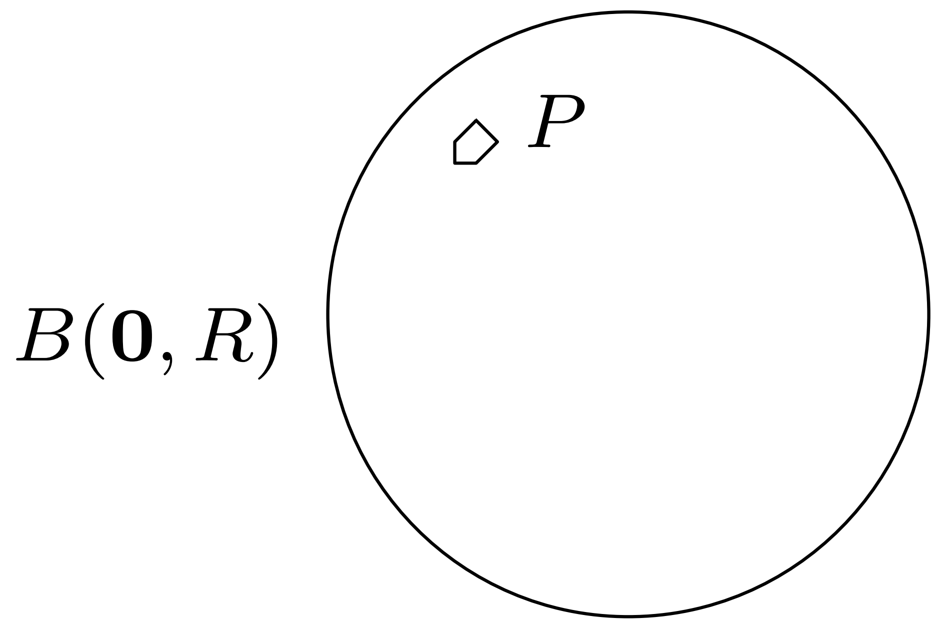
Aside: Also, $Ax \leq b$ has a solution $\iff$

$$Ax \leq b, -2^{\varphi} \leq x_1 \leq 2^{\varphi}, \ldots, -2^{\varphi} \leq x_n \leq 2^{\varphi} \text{ does.}$$

All solutions to the latter live in $B(\mathbf{0}, 2^{\varphi}\sqrt{n})$.

Ellipsoid Algorithm

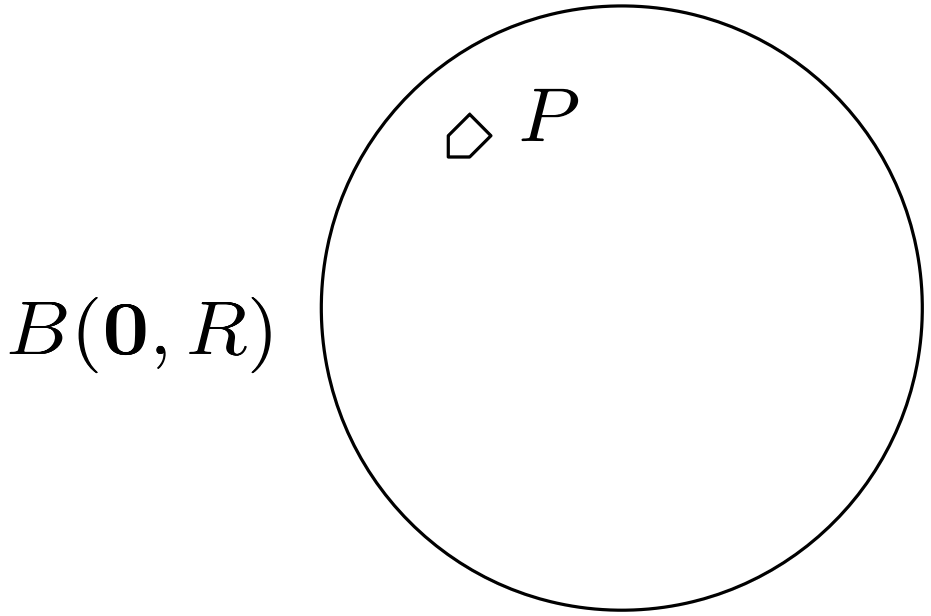
Inputs Matrix A, vector b,
rational numbers $R > \varepsilon > 0$ with feasible region
$P = \{x \in \mathbb{R}^n | Ax \leq b\}$ contained in $B(\mathbf{0}, R)$.



Ellipsoid Algorithm

Inputs Matrix A, vector b,
rational numbers $R > \varepsilon > 0$ with feasible region
$P = \{x \in \mathbb{R}^n | Ax \leq b\}$ contained in $B(\mathbf{0}, R)$.

Output If P contains an ε-ball, return any $y \in P$.
If not, return "no solution" or any $y \in P$.

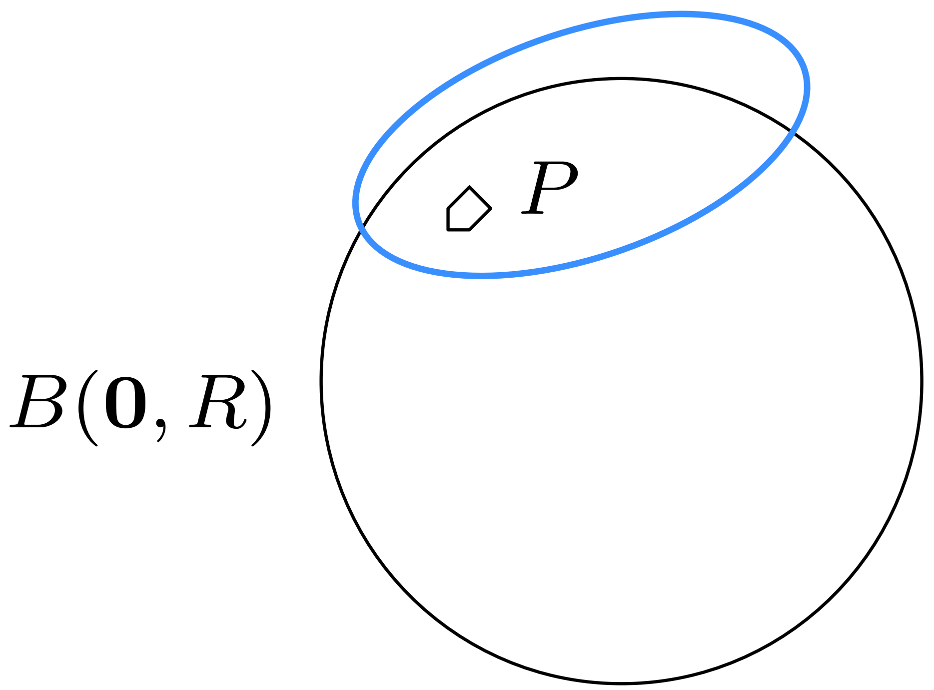


Ellipsoid Algorithm

Inputs Matrix A, vector b,
rational numbers $R > \varepsilon > 0$ with feasible region
$P = \{x \in \mathbb{R}^n | Ax \leq b\}$ contained in $B(\mathbf{0}, R)$.

Output If P contains an ε-ball, return any $y \in P$.
If not, return "no solution" or any $y \in P$.

We will generate ellipsoids
$B(\mathbf{0}, R) = E_0, E_1, E_2, E_3, \ldots$
with $P \subseteq E_k$ for all k.

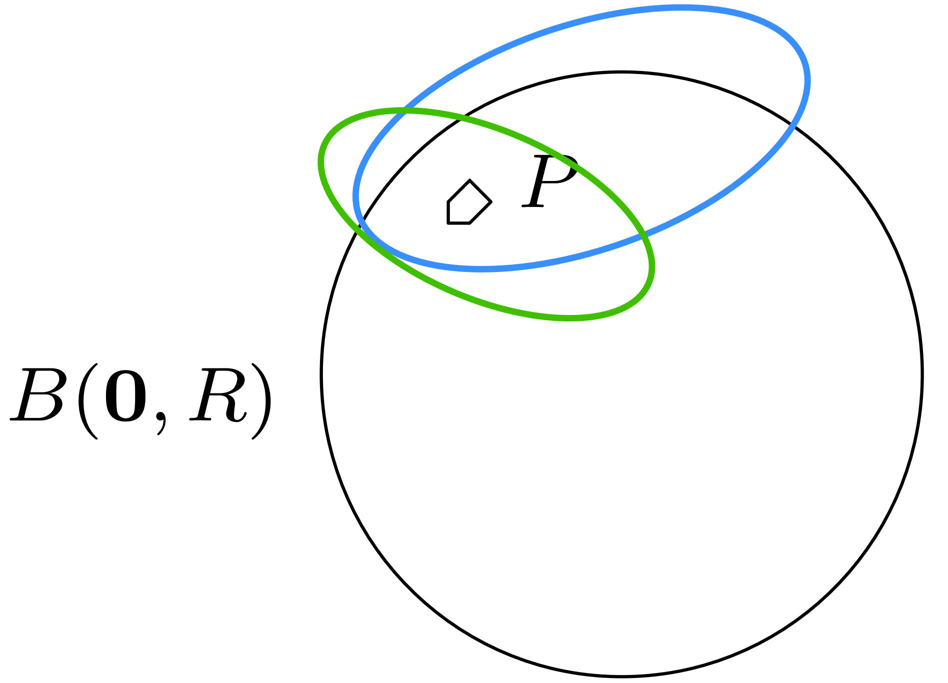


Ellipsoid Algorithm

Inputs Matrix A, vector b,
rational numbers $R > \varepsilon > 0$ with feasible region
$P = \{x \in \mathbb{R}^n | Ax \leq b\}$ contained in $B(\mathbf{0}, R)$.

Output If P contains an ε-ball, return any $y \in P$.
If not, return "no solution" or any $y \in P$.

We will generate ellipsoids
$B(\mathbf{0}, R) = E_0, E_1, E_2, E_3, \ldots$
with $P \subseteq E_k$ for all k.

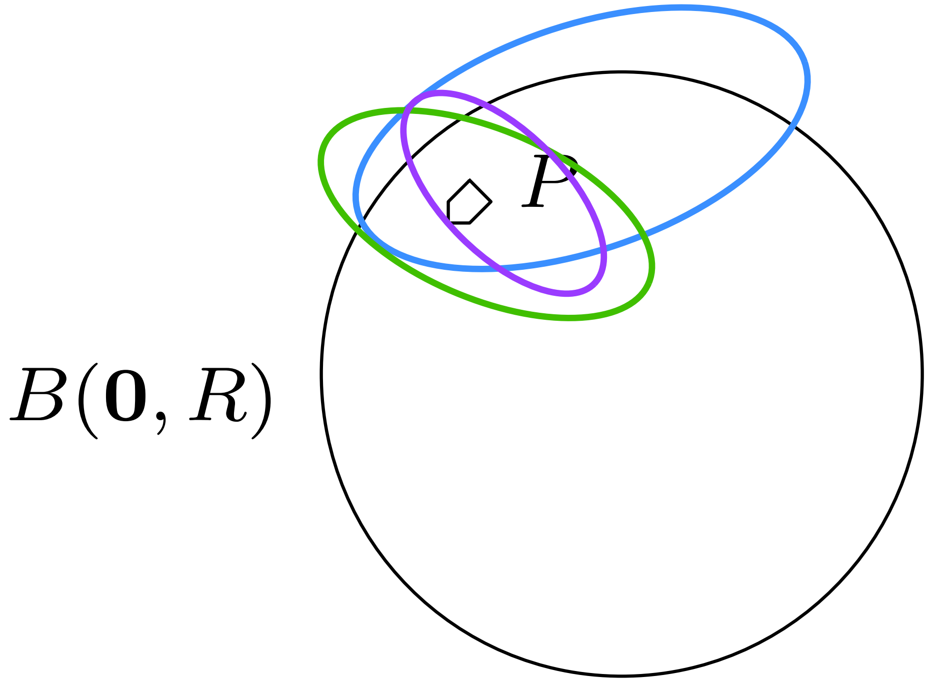


Ellipsoid Algorithm

Inputs Matrix A, vector b,
rational numbers $R > \varepsilon > 0$ with feasible region
$P = \{x \in \mathbb{R}^n | Ax \leq b\}$ contained in $B(\mathbf{0}, R)$.

Output If P contains an ε-ball, return any $y \in P$.
If not, return "no solution" or any $y \in P$.

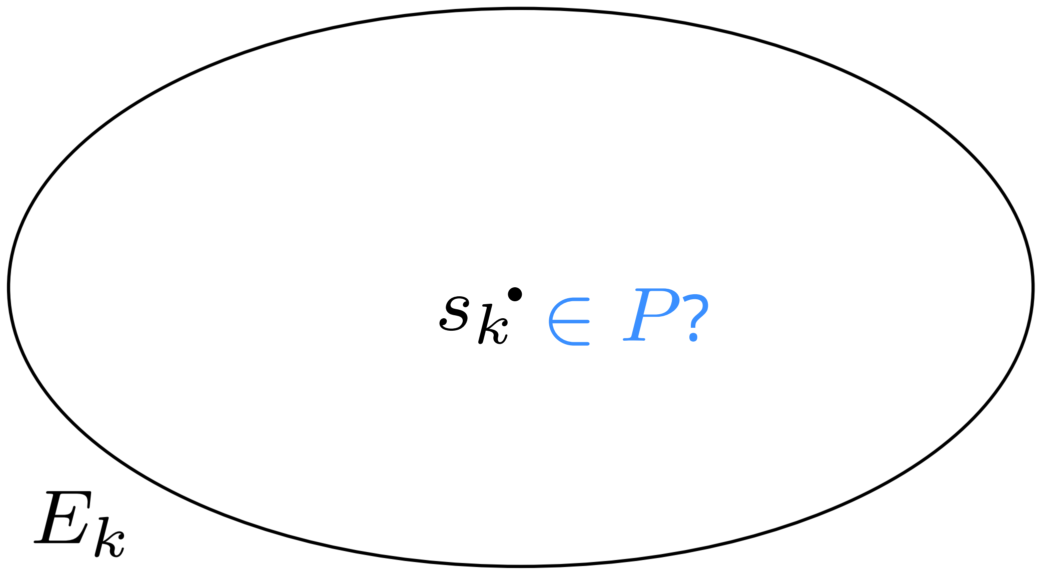
We will generate ellipsoids
$B(\mathbf{0}, R) = E_0, E_1, E_2, E_3, \ldots$
with $P \subseteq E_k$ for all k.



Ellipsoid Algorithm

Iterative step:

If the current ellipsoid E_k has center s_k in P, return that center and stop!
Else, find E_{k+1} from E_k as follows:



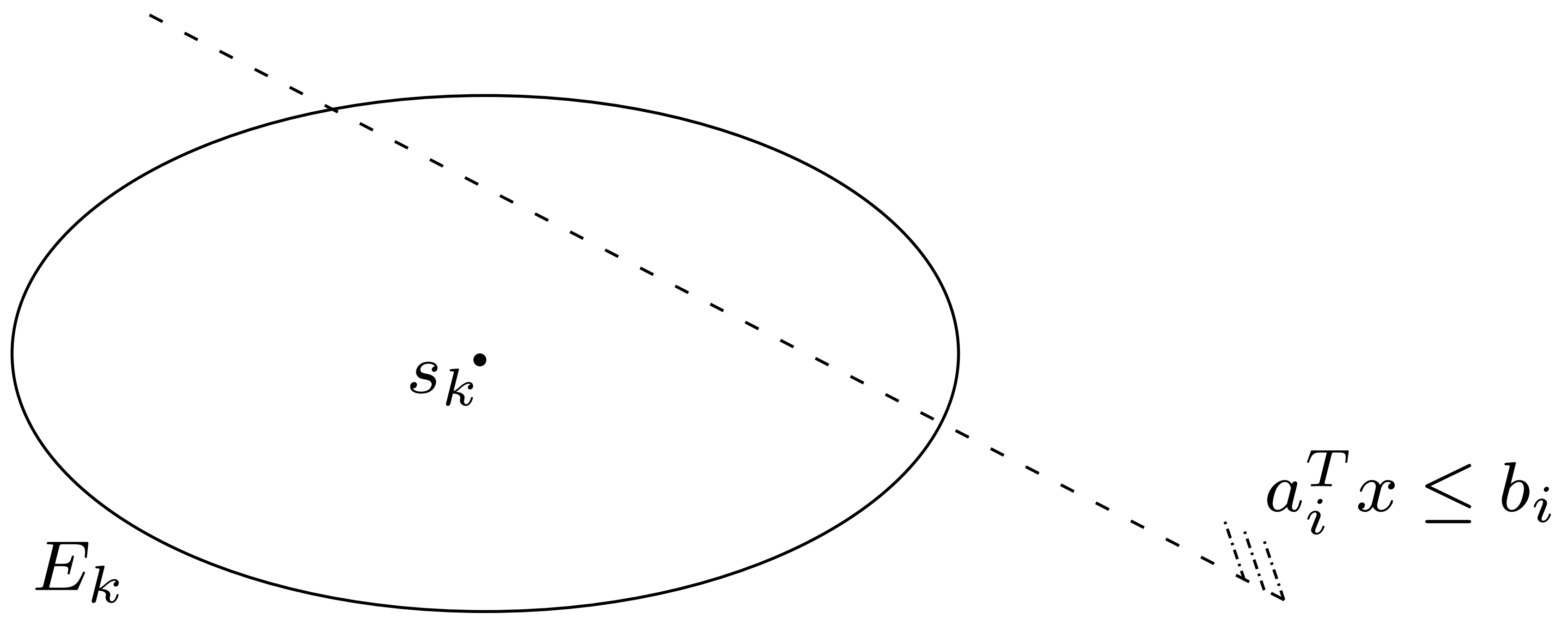
Stop if the volume of E_{k+1} is ever smaller than that of an ε-ball.

Ellipsoid Algorithm

Iterative step:

If the current ellipsoid E_k has center s_k in P, return that center and stop!

Else, find E_{k+1} from E_k as follows:



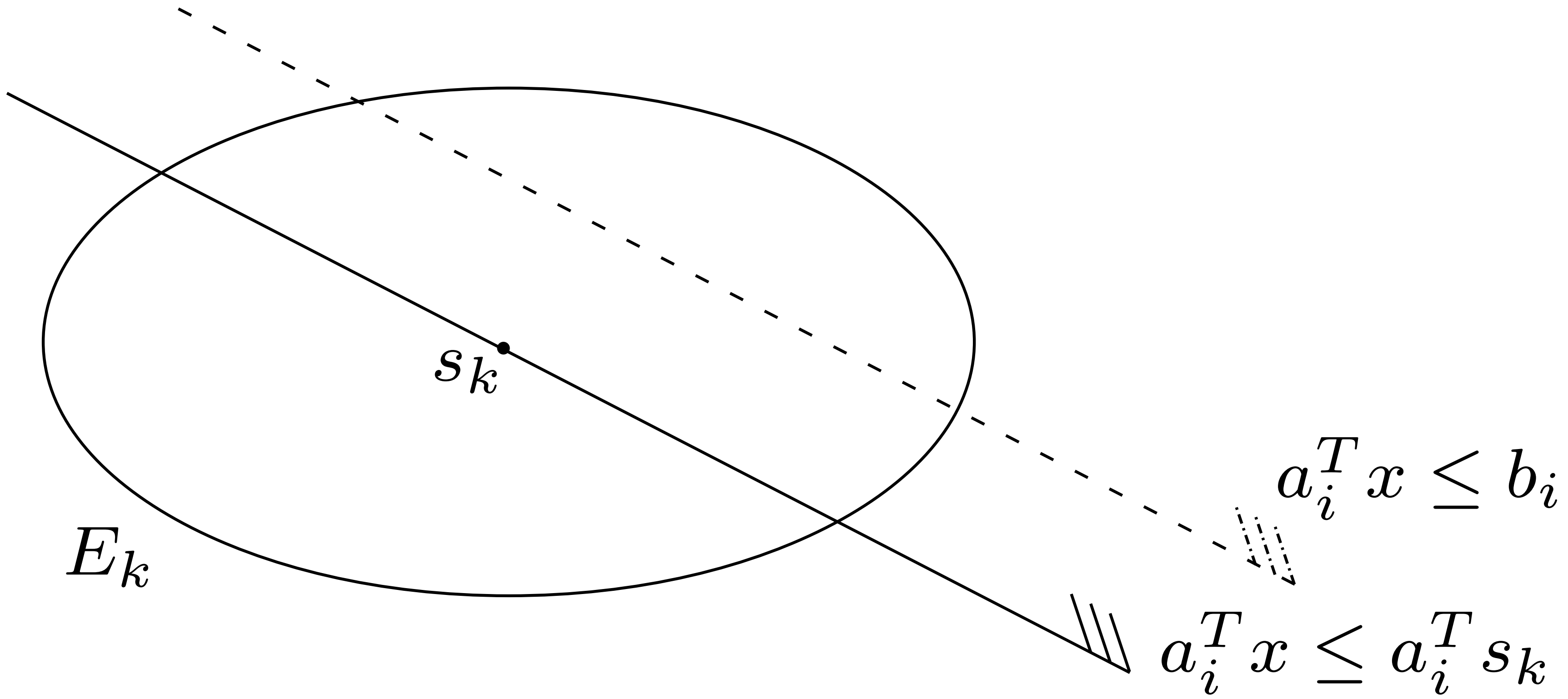
Stop if the volume of E_{k+1} is ever smaller than that of an ε-ball.

Ellipsoid Algorithm

Iterative step:

If the current ellipsoid E_k has center s_k in P, return that center and stop!

Else, find E_{k+1} from E_k as follows:

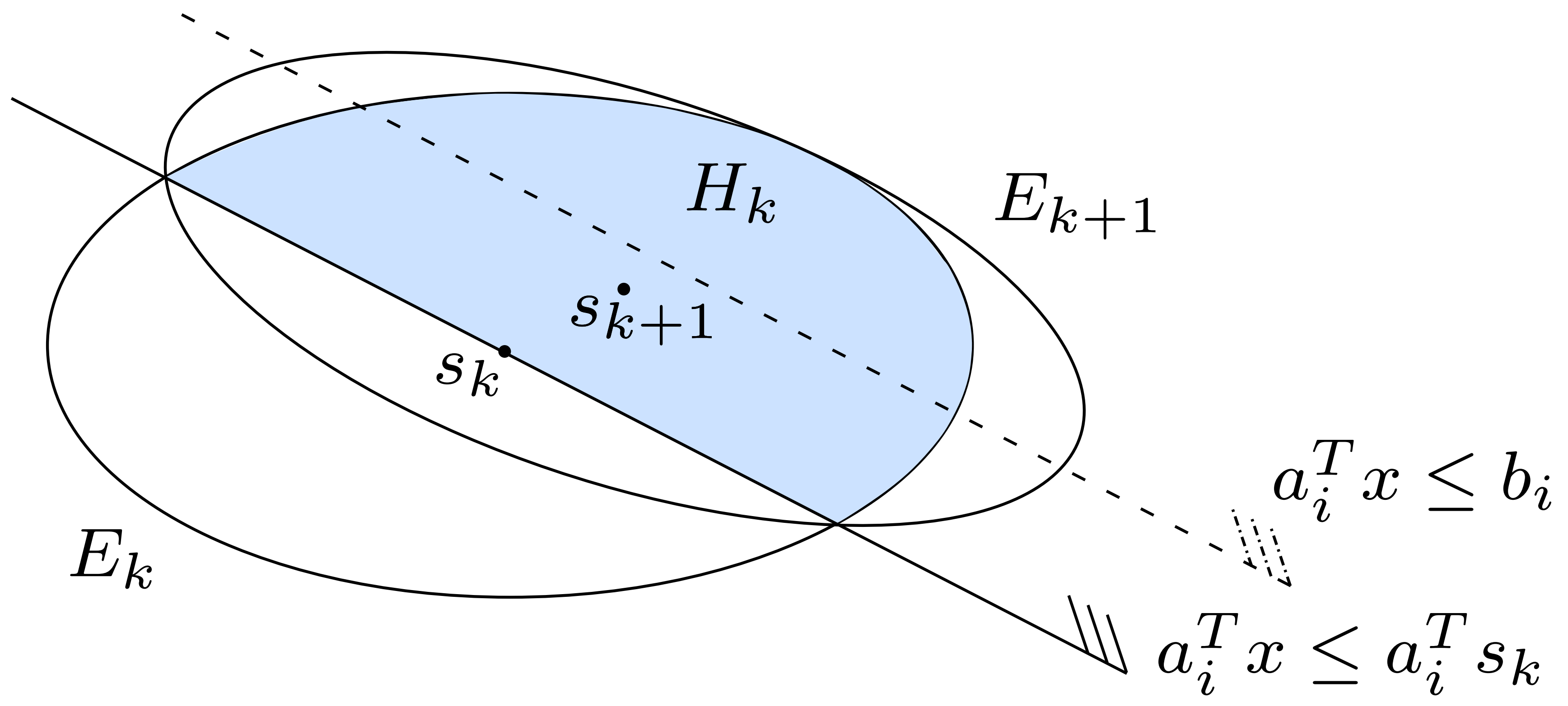


Stop if the volume of E_{k+1} is ever smaller than that of an ε-ball.

Ellipsoid Algorithm

Iterative step:

If the current ellipsoid E_k has center s_k in P, return that center and stop!
Else, find E_{k+1} from E_k as follows:



Stop if the volume of E_{k+1} is ever smaller than that of an ε-ball.

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon)\rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon)\rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

n here because for $n - dim.$ disk of rad. r: volume grows proportional to r^n

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon)\rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

k s.t. $Re^{-k/n(2n+2)} < \varepsilon$

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon)\rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

k s.t. $Re^{-k/n(2n+2)} < \varepsilon$

Subtleties:

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon)\rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

k s.t. $Re^{-k/n(2n+2)} < \varepsilon$

Subtleties:

- Don't compute E_{k+1} exactly (square roots);
  Use rational parameters and expand E_{k+1} slightly.

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon)\rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

k s.t. $Re^{-k/n(2n+2)} < \varepsilon$

Subtleties:

- Don't compute E_{k+1} exactly (square roots);
  Use rational parameters and expand E_{k+1} slightly.
- If the same inequality is violated too many times in a row,
  then the ellipsoids becomes too long.

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon)\rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

k s.t. $Re^{-k/n(2n+2)} < \varepsilon$

Subtleties:

- Don't compute E_{k+1} exactly (square roots);
  Use rational parameters and expand E_{k+1} slightly.
- If the same inequality is violated too many times in a row,
  then the ellipsoids becomes too long.
- Ellipsoids could be replaced with other "rich-enough" families of convex sets,
  like simplices.

Ellipsoid Algorithm

Algorithm analysis: Why is it polynomial?

For all k, $\frac{volume(E_{k+1})}{volume(E_k)} \leq e^{-1/(2n+2)}$,

Hence after $\lceil n(2n+2)\ln(R/\varepsilon) \rceil$ iterations, the volume of E_k is smaller than that of an ε-ball.

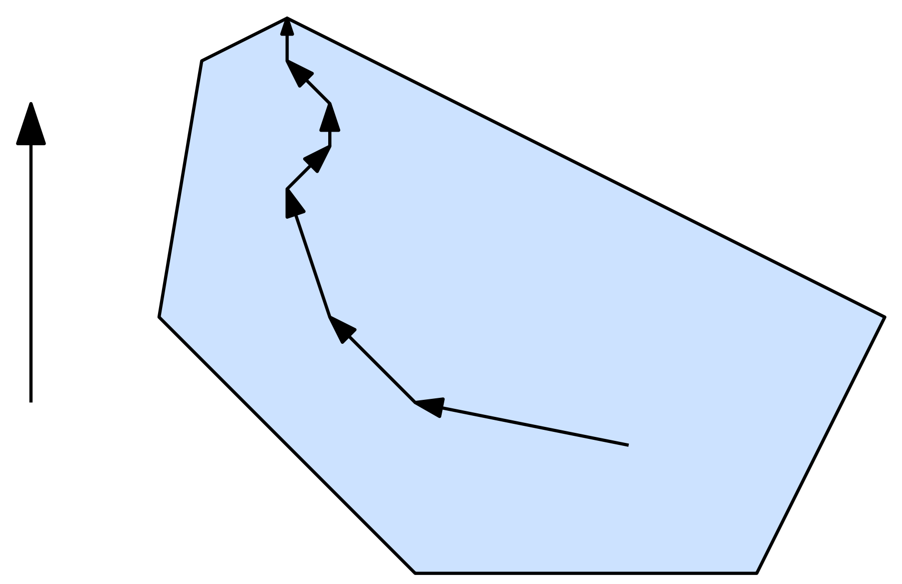
k s.t. $Re^{-k/n(2n+2)} < \varepsilon$

Subtleties:

- Don't compute E_{k+1} exactly (square roots);
  Use rational parameters and expand E_{k+1} slightly.
- If the same inequality is violated too many times in a row,
  then the ellipsoids becomes too long.
- Ellipsoids could be replaced with other "rich-enough" families of convex sets,
  like simplices.
- All we need is a "separation oracle", we don't need to know $Ax \leq b$.
  Could even have infinitely many constraints ($\rightarrow$ semidefinite programming)

Interior Point Methods

for Linear Programming



Interior point methods

- Used for nonlinear problems since 1950’s

- Tested, without success, on linear problems in 1970’s

Interior point methods

- Used for nonlinear problems since 1950’s
- Tested, without success, on linear problems in 1970’s
- Press headlines in 1984: Narendra Karmarkar, IBM, proof of polynomial time on linear problems


The New York Times

AMERICANS IN POLL VIEW GOVERNMENT MORE CONFIDENTLY

TREASURY MAY ASK INCREASES IN TAXES OF SOME CONCERNS

Albany Leaders Predicting a Cut In Income Taxes

Cocaine Traffickers Kill 17 in Peru Raid On Antidrug Team

Vote Comes to a 'Homeland,' But African Problems Linger

Breakthrough in Problem Solving

Homeless Spend Nights in City Welfare Office

High Cuban Defector Speaks Out, Denouncing Castro as 'Impulsive'

Interior point methods

- Used for nonlinear problems since 1950’s

- Tested, without success, on linear problems in 1970’s

- Press headlines in 1984: Narendra Karmarkar, IBM, proof of polynomial time on linear problems


"All the News That's Fit to Print"
The New York Times

AMERICANS IN POLL VIEW GOVERNMENT MORE CONFIDENTLY

TREASURY MAY ASK INCREASES IN TAXES OF SOME CONCERNS

Albany Leaders Predicting a Cut In Income Taxes

Cocaine Traffickers Kill 17 in Peru Raid On Antidrug Team

Vote Comes to a 'Homeland,' But African Problems Linger

Breakthrough in Problem Solving

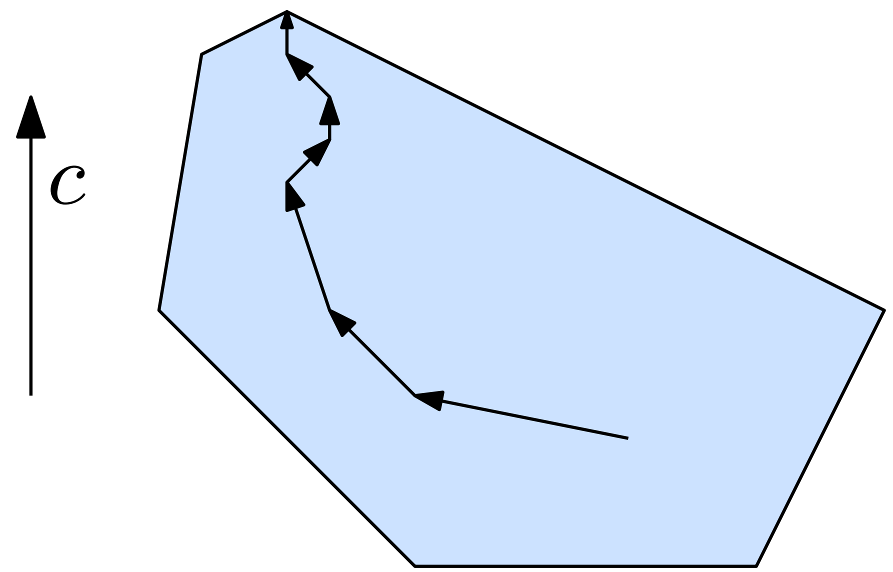
Homeless Spend Nights in City Welfare Office

High Cuban Defector Speaks Out, Denouncing Castro as 'Impulsive'

- Now competitive with the simplex method, especially on large problems (too large for the 1970’s): Rely on powerful routines for sparse system of equations.

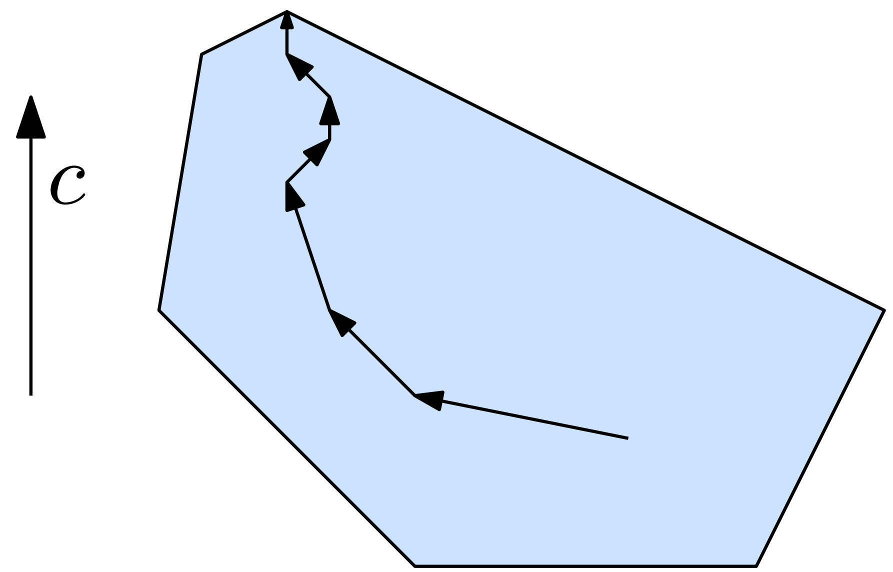
Interior point methods

- Interior methods walk along the interior, until they hop to an exact optimum via a last rounding step.



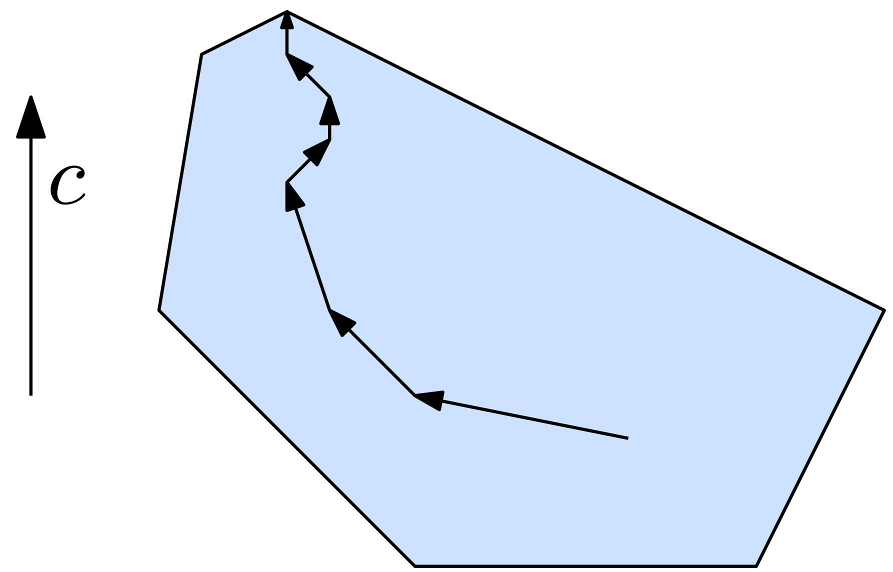
Interior point methods

- Interior methods walk along the interior, until they hop to an exact optimum via a last rounding step.
- By contrast, the simplex method walks along the boundary, and the ellipsoid method encircles the set of feasible solutions from the outside.



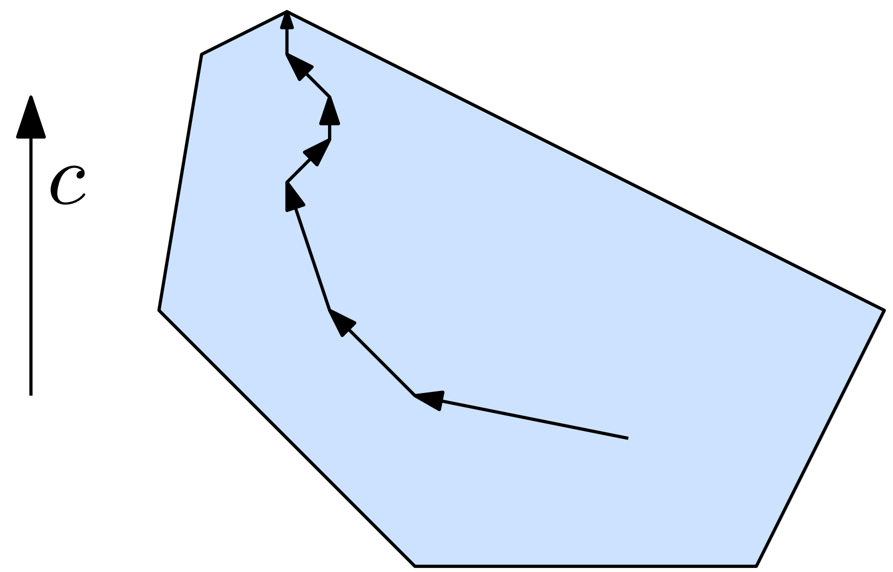
Interior point methods

- Interior methods walk along the interior, until they hop to an exact optimum via a last rounding step.
- By contrast, the simplex method walks along the boundary, and the ellipsoid method encircles the set of feasible solutions from the outside.
- Avoids combinatorial intricacies of the boundary.



Interior point methods

- Interior methods walk along the interior, until they hop to an exact optimum via a last rounding step.
- By contrast, the simplex method walks along the boundary, and the ellipsoid method encircles the set of feasible solutions from the outside.
- Avoids combinatorial intricacies of the boundary.



Types of interior point methods:

Central path, potential reduction, affine scaling.

For each of them:

primal, dual, primal-dual, or self-dual.

The Central Path

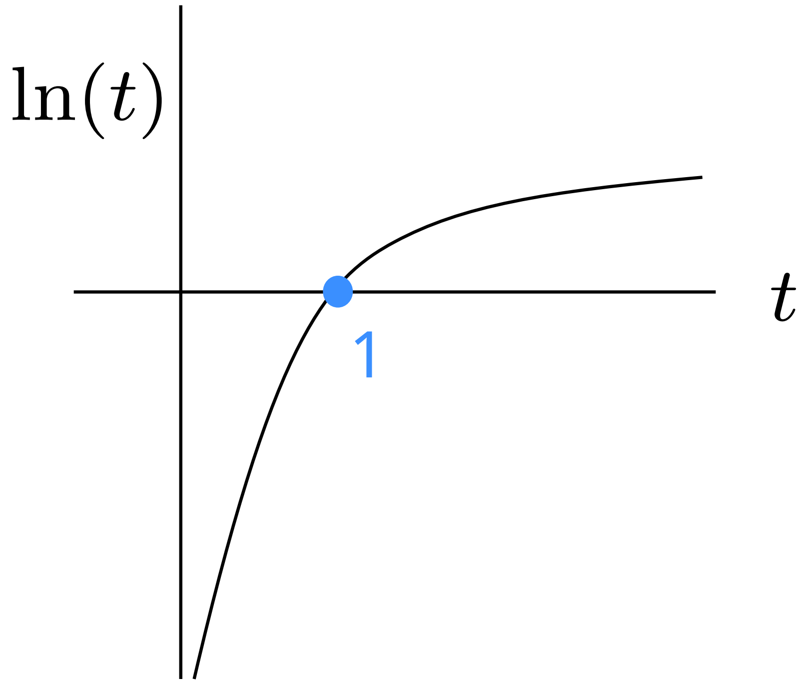
Maximize $c^T x$ subject to $Ax \leq b$

For $\mu > 0$, define

$$f_\mu(x) = c^T x + \mu \sum_{i=1}^{m} \ln(b_i - a_i x)$$

where a_i is the i-th row of A.

Logarithmic barrier



The Central Path

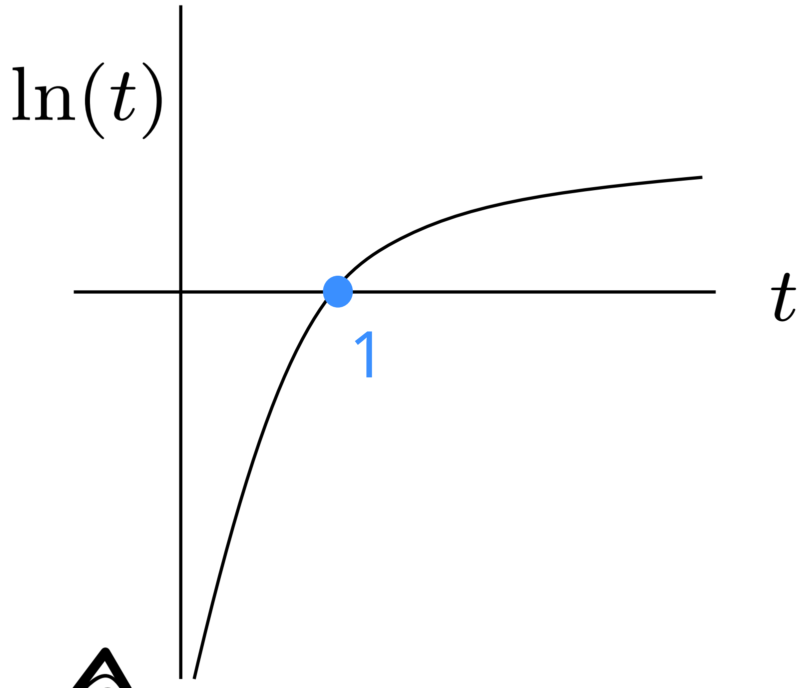
Maximize $c^T x$ subject to $Ax \le b$

For $\mu > 0$, define

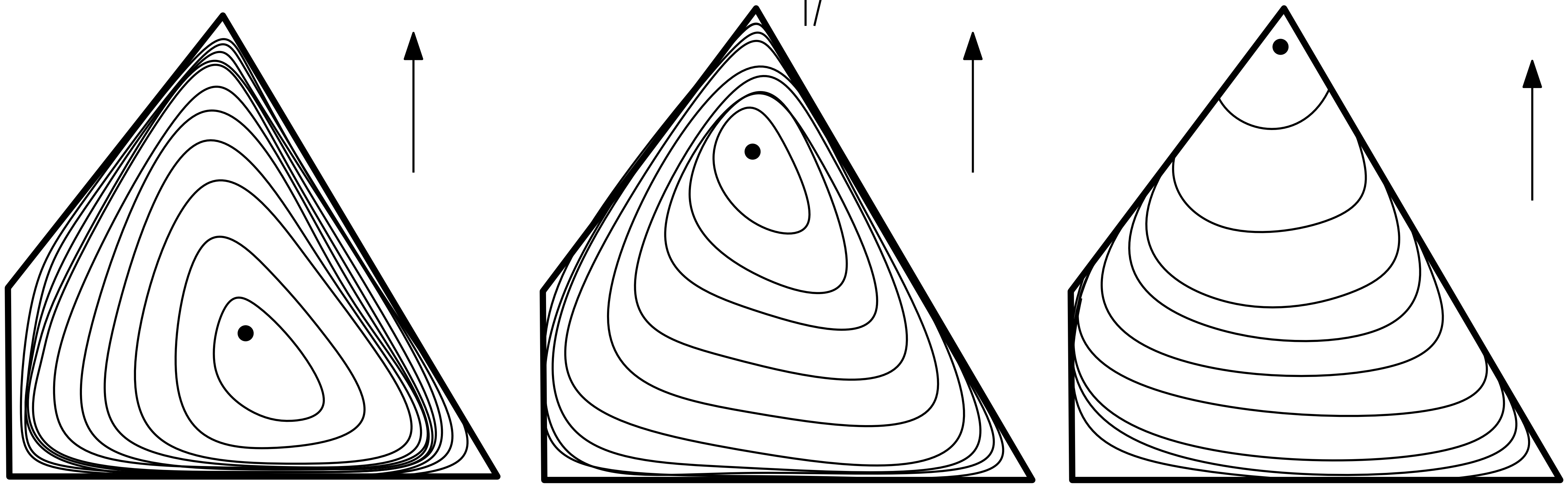
$$f_\mu(x) = c^T x + \mu \sum_{i=1}^{m} \ln(b_i - a_i x)$$

where a_i is the i-th row of A.

Logarithmic barrier

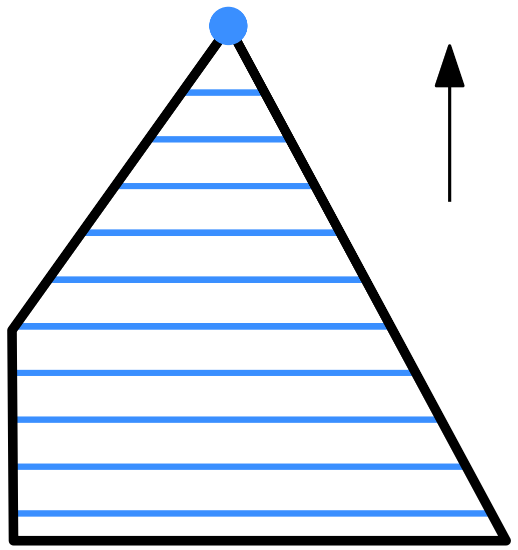


"furthest point
from boundary"



$\mu = 100$ $\mu = 1$ $\mu = 1/100$

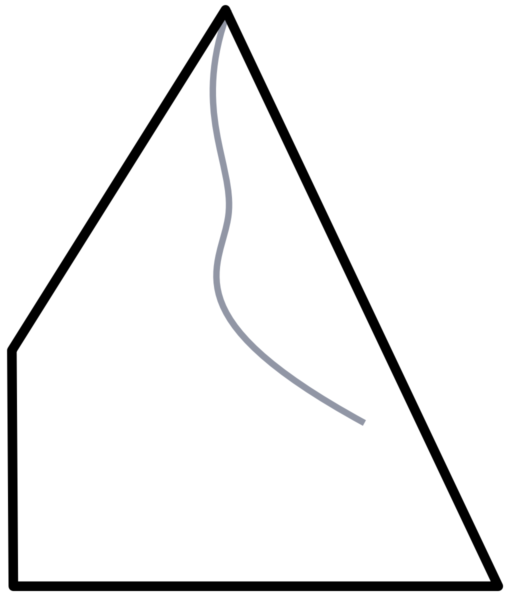
The Central Path



$\mu = 0$ (the limit)

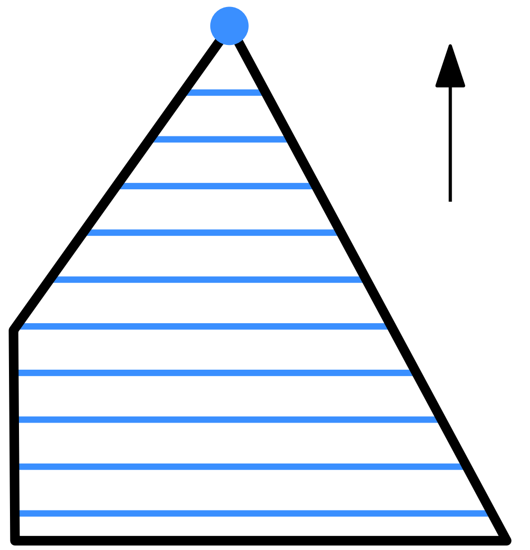
Let $x^*(\mu)$ be the unique solution to
Maximize $f_\mu(x)$subject to $Ax \leq b$.

redundant



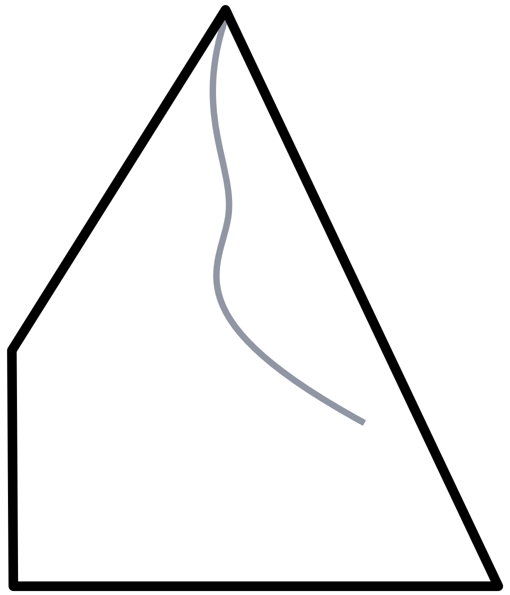
Def: The central path is the curve $x^*(\mu)$ for $\mu > 0$.

The Central Path

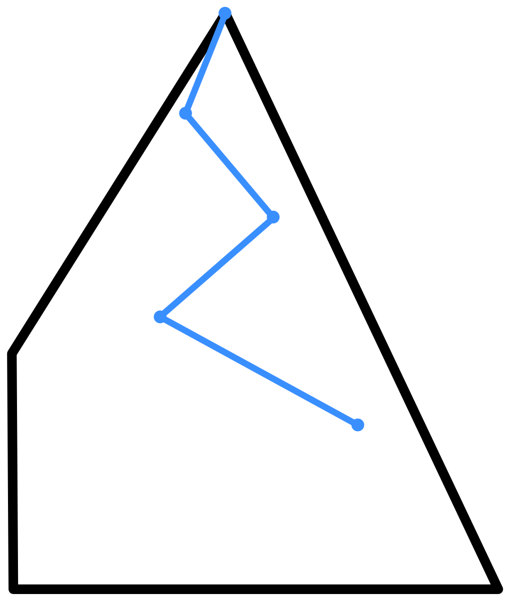


Let $x^*(\mu)$ be the unique solution to
Maximize $f_\mu(x)$subject to $Ax \leq b$.

redundant



Def: The central path is the curve $x^*(\mu)$ for $\mu > 0$.

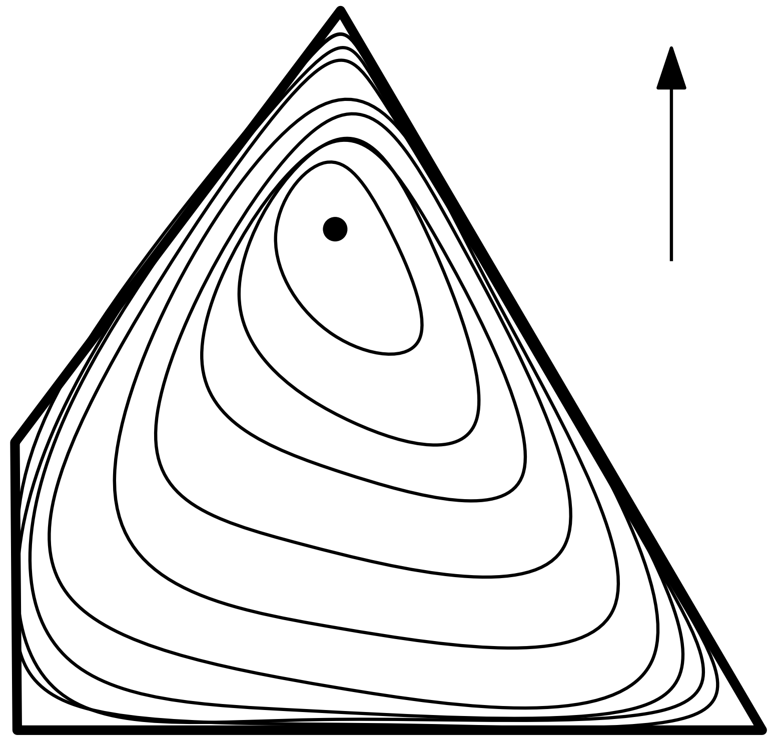


Actual algorithms will only approximately follow this path.

The Central Path

For $\mu > 0$ fixed, why is there a unique optimum $x^*(\mu)$?

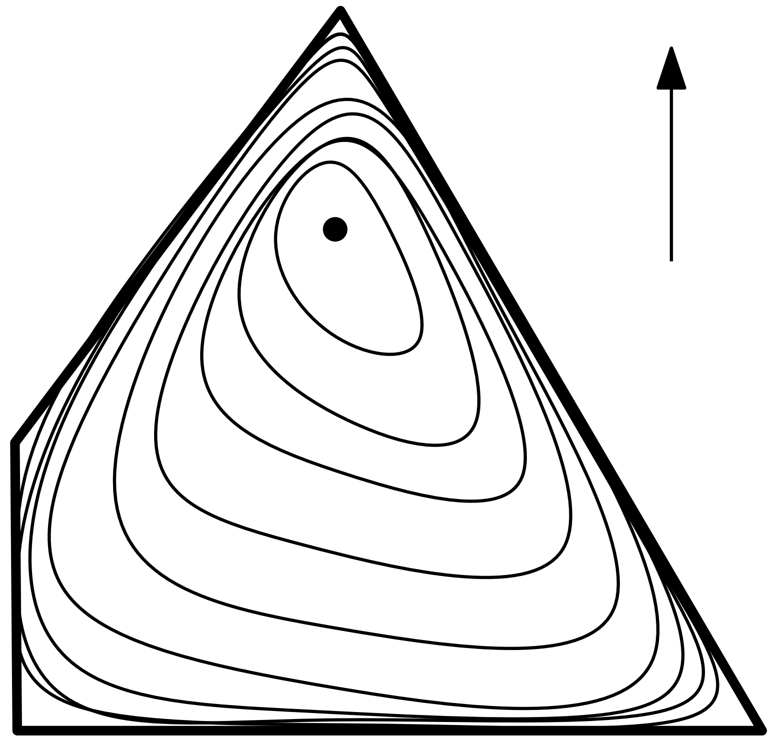
- An optimum exists since we have a continous function
$$f_\mu(x) = c^T x + \mu \sum_i \ln(b_i - a_i x)$$
on a compact set
$\{x : Ax \leq b$ and $f_\mu(x) \geq f_\mu(y)\}$.



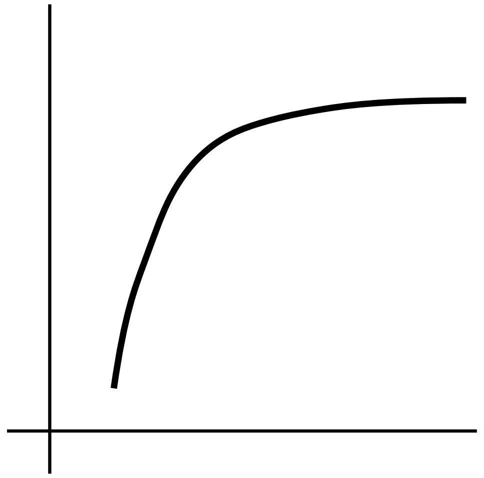
The Central Path

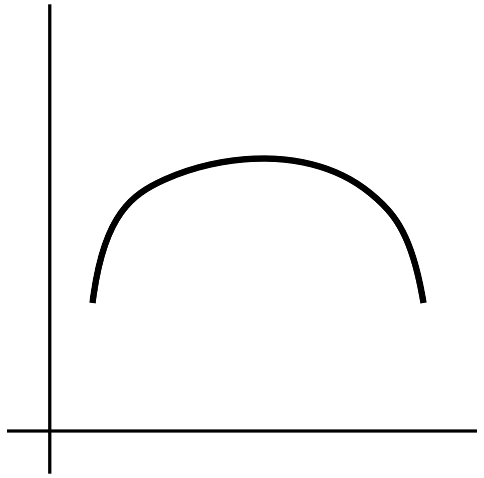
For $\mu > 0$ fixed, why is there a unique optimum $x^*(\mu)$?

- An optimum exists since we have a continous function
$$f_\mu(x) = c^T x + \mu \sum_i \ln(b_i - a_i x)$$
on a compact set
$\{x : Ax \leq b$ and $f_\mu(x) \geq f_\mu(y)\}$.



- The optimum is unique since f_μ is strictly concave for $\mu > 0$.





The primal-dual Central Path (equational form)

Maximize $c^T x$ subject to $Ax = b,\ x \geq 0$

size $m \times n$

$$f_\mu(x) = c^T x + \mu \sum_{j=1}^{n} \ln(x_j)$$

The primal-dual Central Path (equational form)

Maximize $c^T x$ subject to $Ax = b,\ x \geq 0$

size $m \times n$

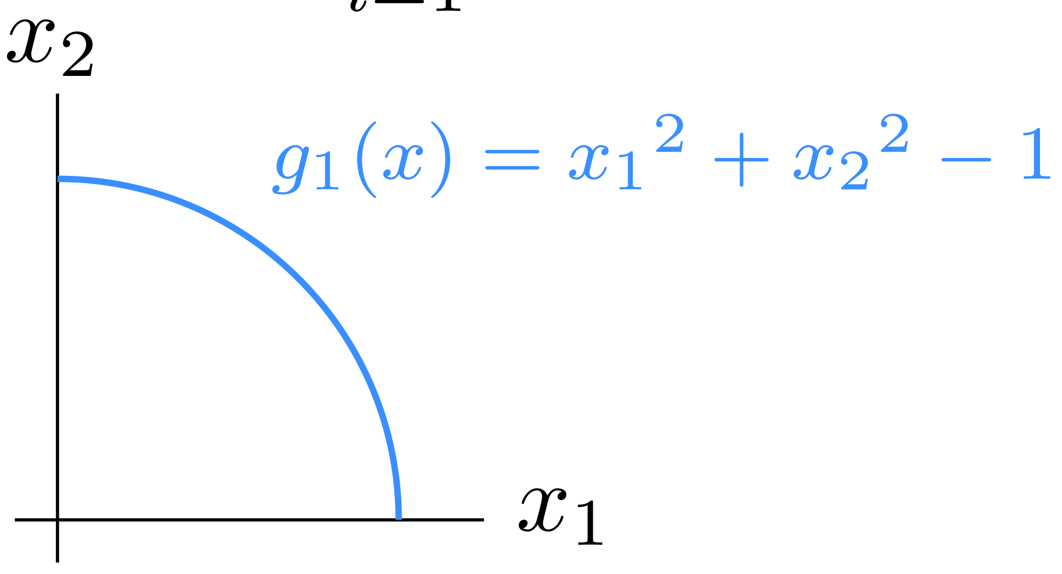
$$f_\mu(x) = c^T x + \mu \sum_{j=1}^{n} \ln(x_j)$$

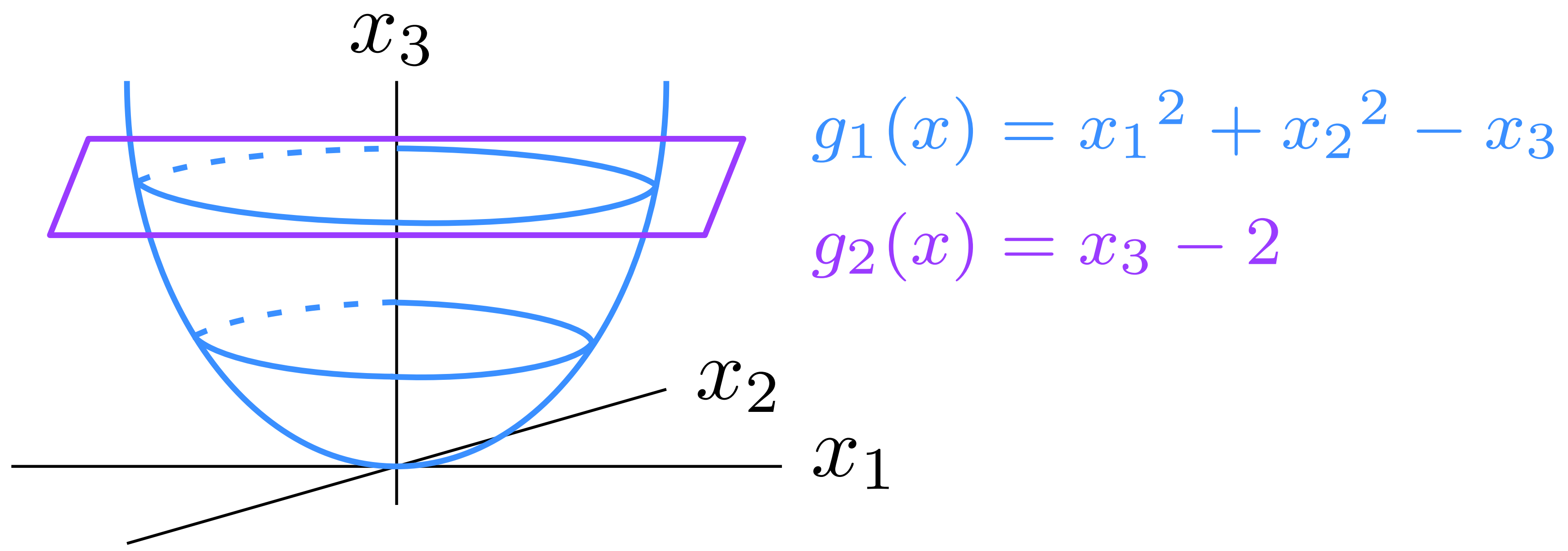
Lagrange Multipliers:

A maximum of $f(x)$ subject to $g_1(x) = g_2(x) = \ldots = g_m(x) = 0$ satisfies

$$\nabla f(x) = \sum_{i=1}^{m} y_i \nabla g_i(x)$$

(Functions are $\mathbb{R}^n \to \mathbb{R}$, gradients are row vectors.)





The primal-dual Central Path (equational form)

Apply Lagrange multipliers to

$$g_i(x) = a_i x - b_i$$

Maximize $\quad f_\mu(x) = c^T x + \mu \sum_{j=1}^{m} \ln(x_j) \quad$ subject to $\; Ax = b, \; x \geq 0$

to get $\quad c + \mu \left(\frac{1}{x_1}, \ldots, \frac{1}{x_n} \right) = \nabla f(x) = \sum_{i=1}^{m} y_i \nabla g_i(x) = \sum_{i=1}^{m} y_i a_i = A^T y$

The primal-dual Central Path (equational form)

Apply Lagrange multipliers to

$$g_i(x) = a_i x - b_i$$

Maximize $\quad f_\mu(x) = c^T x + \mu \sum_{j=1}^{m} \ln(x_j) \quad$ subject to $\; Ax = b,\; x \geq 0$

to get $\quad c + \mu \left(\frac{1}{x_1}, \ldots, \frac{1}{x_n} \right) = \nabla f(x) = \sum_{i=1}^{m} y_i \nabla g_i(x) = \sum_{i=1}^{m} y_i a_i = A^T y$

Introduce the nonnegative vector $s = \mu \left(\frac{1}{x_1}, \ldots, \frac{1}{x_n} \right)$ to get

$$\begin{gathered} Ax = b \\ A^T y - s = c \\ (s_1 x_1, s_2 x_2, \ldots, s_n x_n) = (\mu, \mu, \ldots, \mu) \qquad \text{(Not linear)} \\ x, s \geq 0 \qquad y \in \mathbb{R}^m \end{gathered}$$

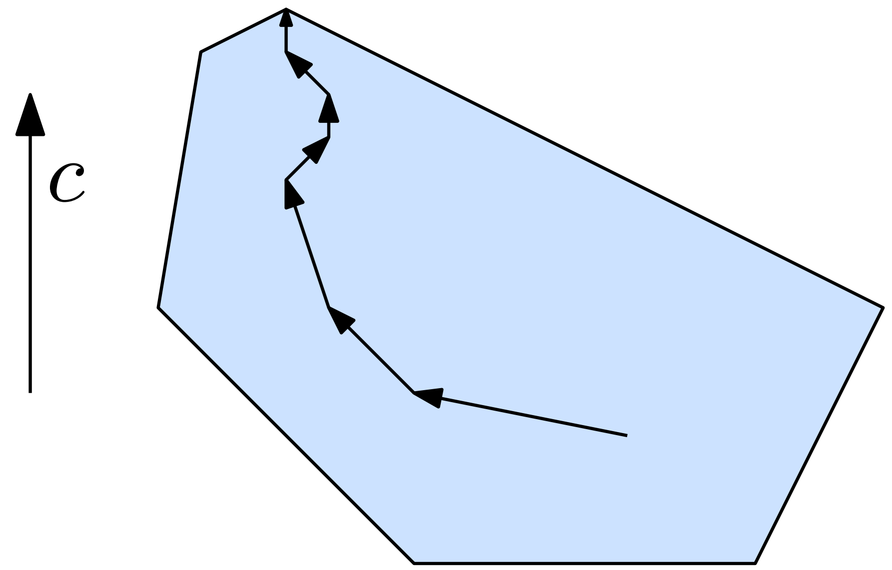
The primal-dual Central Path (equational form)

Introduce the nonnegative vector $s = \mu \left(\frac{1}{x_1}, \ldots, \frac{1}{x_n} \right)$ to get

$$\begin{aligned} Ax &= b \\ A^T y - s &= c \\ (s_1 x_1, s_2 x_2, \ldots, s_n x_n) &= (\mu, \mu, \ldots, \mu) \qquad \text{(Not linear)} \\ x, s &\geq 0 \qquad y \in \mathbb{R}^m \end{aligned}$$

The primal-dual central path is

$$\left\{ (x^*(\mu), y^*(\mu), s^*(\mu)) \in \mathbb{R}^{2n+m} : \mu > 0 \right\}$$



The primal-dual Central Path (equational form)

In some sense, Lagrange multipliers recover the duality of linear programming!

We started with

Maximize $c^T x$ subject to $Ax = b,\ x \geq 0$

whose dual is

Minimize $b^T y$ subject to $A^T y \geq c,\ y \in \mathbb{R}^m$.

The primal-dual Central Path (equational form)

In some sense, Lagrange multipliers recover the duality of linear programming!

We started with

Maximize $c^T x$ subject to $Ax = b,\ x \geq 0$

whose dual is

Minimize $b^T y$ subject to $A^T y \geq c,\ y \in \mathbb{R}^m$.

We derived

$$\boxed{\begin{array}{rcll} Ax &=& b & \\ A^T y - s &=& c & \\ (s_1 x_1, s_2 x_2, \ldots, s_n x_n) &=& (\mu, \mu, \ldots, \mu) & \text{(Not linear)} \\ x, s &\geq& 0 & y \in \mathbb{R}^m \end{array}}$$

for $\mu > 0$, but setting $\mu = 0$ gives $(s_1 x_1, \ldots, s_n x_n) = (0, \ldots, 0)$, i.e.,

$s^T x = 0$ by nonnegativity.

The primal-dual Central Path (equational form)

Setting $\mu = 0$ gives $(s_1 x_1, \ldots, s_n x_n) = (0, \ldots, 0)$, i.e.,

$s^T x = 0$ by nonnegativity.

So
$$\begin{aligned} 0 &= s^T x \\ &= \left(A^T y - c\right)^T x \\ &= y^T A x - c^T x \\ &= y^T b - c^T x \qquad \text{Since } Ax = b \end{aligned}$$

So x is a feasible solution of the primal,

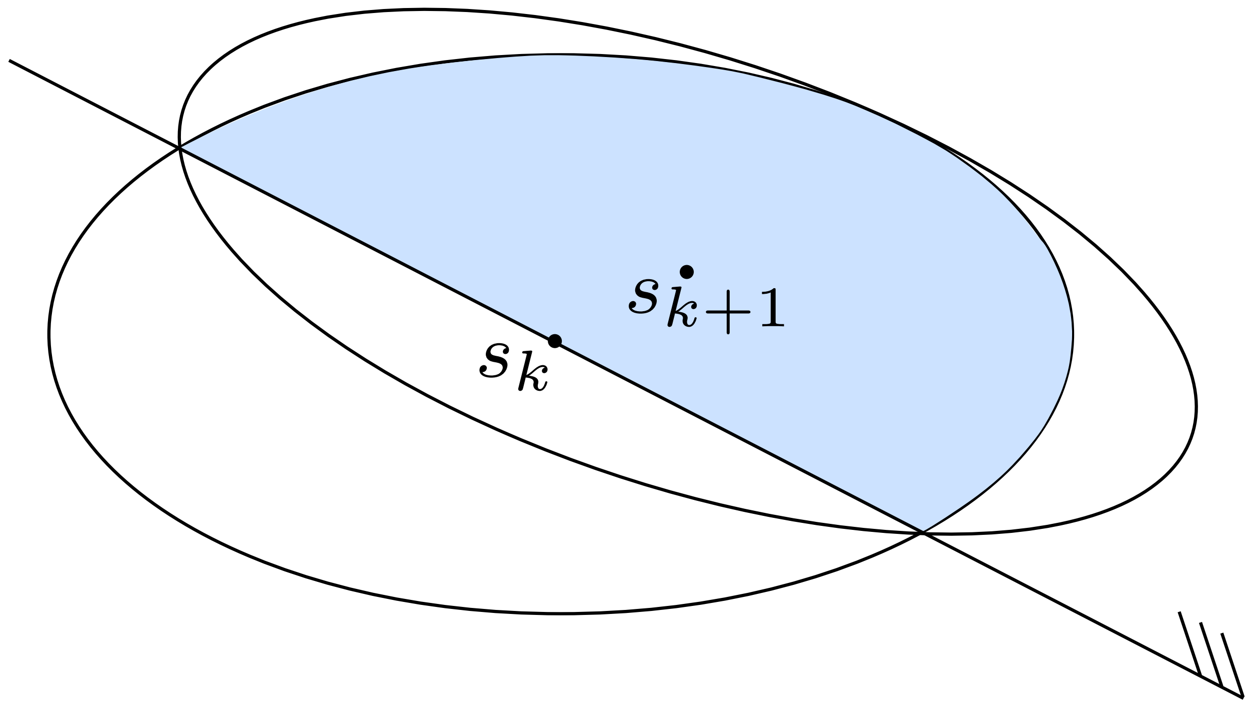
y is a feasible solution of the dual (with slack variables s_j)

and $y^T b = c^T x$ implies these feasible solutions are optimal, by strong duality.

Summary

Ellipsoid method:

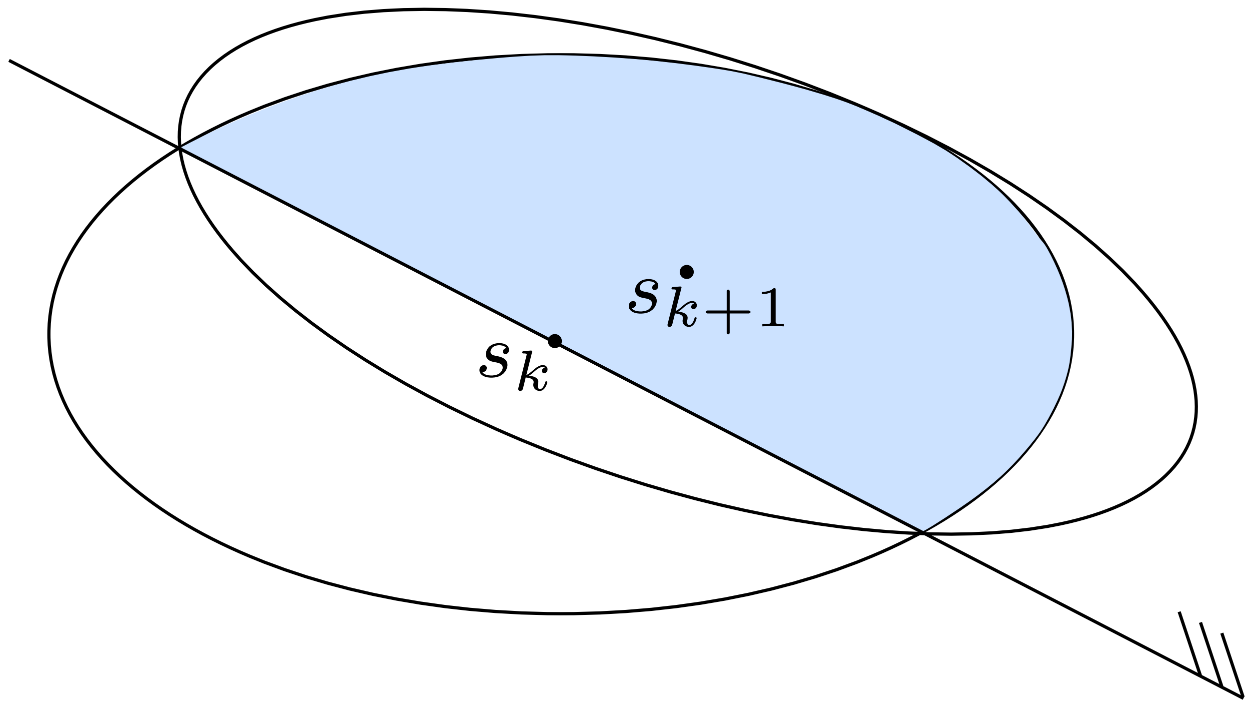
find smaller and smaller ellipsoids containing set of feasible solution inside; cutting by halfspaces



Summary

Ellipsoid method:

find smaller and smaller ellipsoids containing set of feasible solution inside; cutting by halfspaces



Interior point method – central paths:

force solution inside by adding a “barrier” to the objective function. Central path: by lowering effect μ of barrier moves towards LP solution. Uses Lagrange Multipliers.