ORF522 – Linear and Nonlinear Optimization

6. Numerical linear algebra and simplex implementation

Bartolomeo Stellato — Fall 2020

Ed forum

- Can we use a random pivot rule?
  Yes! Sometimes quadratic convergence. Not used in modern solvers though.
- How does the basis of the perturbed problem relate to the basis of the original problem?
  We cannot go easily back to the original problem basis! However, the solution will be very likely close.
- Do certain pivot rules not only avoid cycling but also have nice properties such as polynomial amortized time or some other sort of nice property in a "average" sense?
  Yes, Bland's rule enjoys these properties in the average case.
- What happens with the perturbation approach if the matrix is ill-conditioned?
  Bad things… it could definitely give us a completely wrong solution.
- Any rule to pick M for the big-M method?
  If you keep the data symbolic, a simple rule is to consider M larger than any other number appearing in the algorithm. In this case, whenever it is compared to any number, it is larger. In general, simplex-like methods always work in two-phases and avoid the big-M "tuning" problem.
- Can you just remove the redundant constraints in case of degeneracy?
  Degeneracy for sure depends on the way the polyhedron is represented but it might not be always that easy. It does not only happen in case of redundant constraints (the ones that can be removed without changing the shape of the polytope).
- Is there already some work done on the distribution of complexity for some class of linear programs, say, if we are searching on a probability simplex?
  Yes, there is interesting work about small perturbations and simplex complexity.
- Will the solver always go down the same route in its iterations for a given problem, or will it involve a random seed such that each execution is different? For a given problem, will we encounter cases such that one execution finishes very fast, while a repeated execution may exceed max iterations?
  If the solver is deterministic, it is always the same route. It is usually the case in common solvers.

[Gärtner, B., Henk, M., & Ziegler, G. M. (1998). Randomized simplex algorithms on Klee-Minty cubes. *Combinatorica*, *18*(3), 349-372.]

[Kelner, J. A., & Spielman, D. A. (2006). A randomized polynomial-time simplex algorithm for linear programming. In Proc. of ACM symposium on Theory of computing]

Recap

An iteration of the simplex method

First part

We start with a basic feasible solution x and a basis matrix $B = \begin{bmatrix} A_{B(1)} & \dots, A_{B(m)} \end{bmatrix}$

1. Compute the reduced costs $\bar{c}_j = c_j - c_B^T B^{-1} A_j$ for $j \in N$
2. If $\bar{c_j} \geq 0$, x **optimal**. **break**
3. Choose j such that $\bar{c}_j < 0$

An iteration of the simplex method

Second part

4. Compute search direction components $d_B = -B^{-1}A_j$
5. If $d_B \geq 0$, the problem is **unbounded** and the optimal value is $-\infty$. **break**
6. Compute step length $\theta^\star = \min_{\{i \in B | d_i < 0\}} \left(-\frac{x_i}{d_i} \right)$
7. Define y such that $y = x + \theta^\star d$

Today's agenda

[Chapter 3, Bertsimas and Tsitsiklis]
[Chapter 13, Nocedal and Wright]
[Chapter 8, Vanderbei]

- Numerical linear algebra
- Realistic simplex implementation
- Example
- Empirical complexity

Numerical linear algebra

Deeper look at complexity

Flop count

floating-point operations: one addition, subtraction, multiplication, division

Estimate complexity of an algorithm

- Express number of flops as a **function of problem dimensions**
- Simplify and keep only leading terms

Remarks

- Not accurate in modern computers (multicore, GPU, etc.)
- Still rough and widely-used estimate of complexity

Complexity

Basic examples

Vector operations ($x, y \in \mathbf{R}^n$)

- Inner product $x^T y$: $2n - 1$ flops
- Sum $x + y$ or scalar multiplication αx: n flops

Matrix-vector product ($y = Ax$ with $A \in \mathbf{R}^{m \times n}$)

- $m(2n - 1)$ flops
- $2N$ if A is sparse with N nonzero elements

Matrix-matrix product ($C = AB$ with $A \in \mathbf{R}^{m \times n}, B \in \mathbb{R}^{n \times p}$)

- $pm(2n - 1)$ flops
- Less if A and/or B are sparse

Complexity

Solving linear system

Execution time (cost) of solving $Ax = b$ with $A \in \mathbf{R}^{n \times n}$

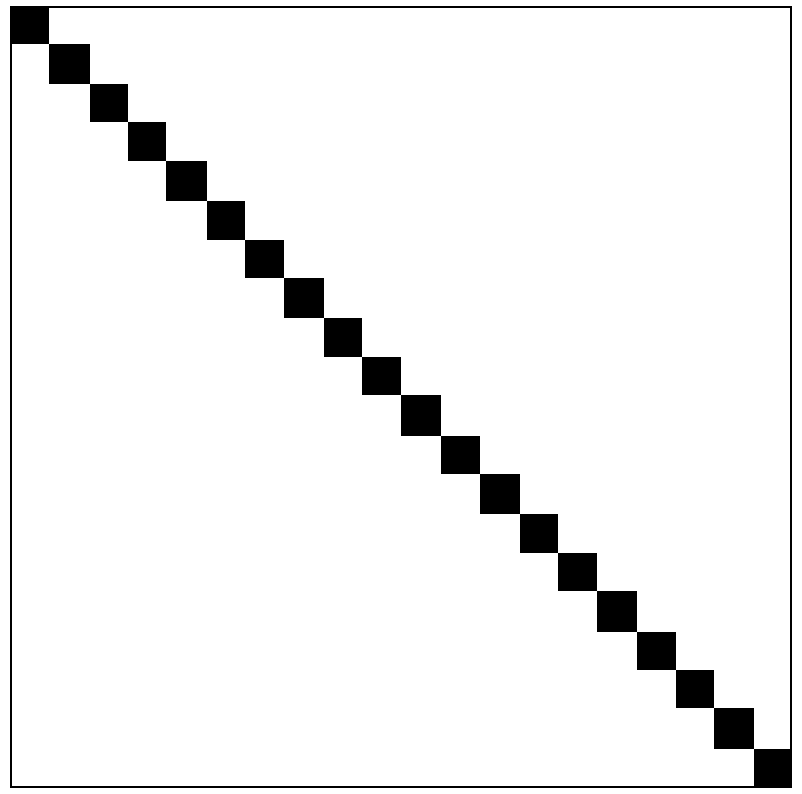
General case $O(n^3)$

Much less if A structured (sparse, banded, Toepliz, etc.)

You (almost) **never compute** A^{-1} explicitly!

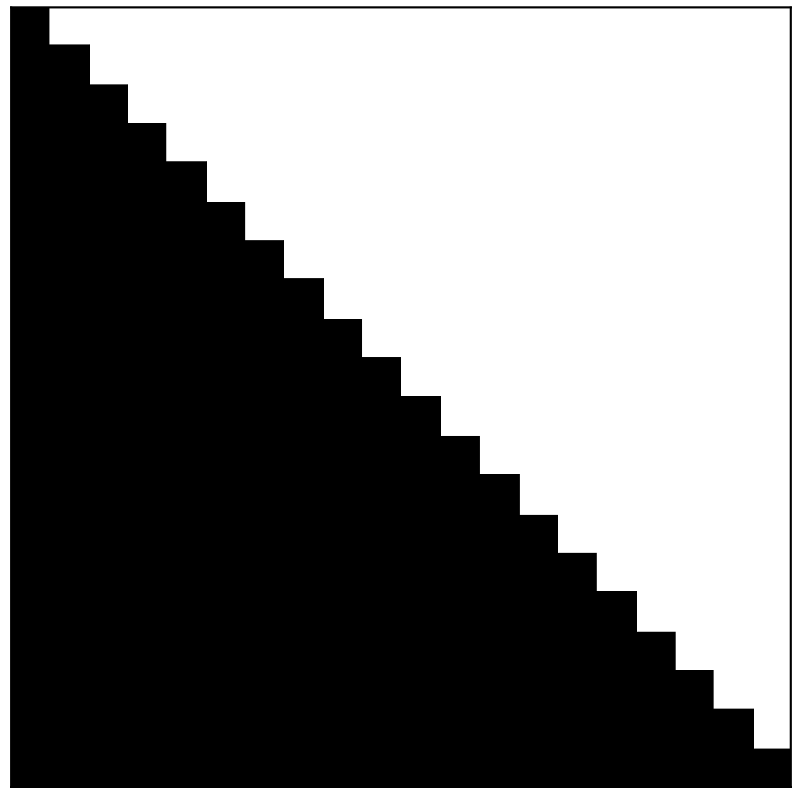
- Numerically unstable (divisions)
- You lose structure

Easy linear systems



Diagonal matrices ($a_{ij} = 0$ if $i \neq j$): $O(n)$ flops

$$x = A^{-1}b = (b_1/a_{11}, \ldots, b_n/a_{nn})$$



Lower-triangular ($a_{ij} = 0$ if $j > i$): $O(n^2)$ flops (**forward substitution**)

$$x_1 = b_1/a_{11}$$

$$x_2 = (b_2 - a_{21}x_1)/a_{22}$$

$$\vdots$$

$$x_n = (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})/a_{nn}$$

Upper-triangular ($a_{ij} = 0$ if $j < i$): $O(n^2)$ flops (**backward substitution**)

Other easy linear systems

Orthogonal matrices ($A^{-1} = A^T$)
$O(n^2)$ flops to compute $x = A^T b$ for general A

Permutation matrices

$$a_{ij} = \begin{cases} 1 & j = \pi_i \\ 0 & \text{otherwise} \end{cases}$$

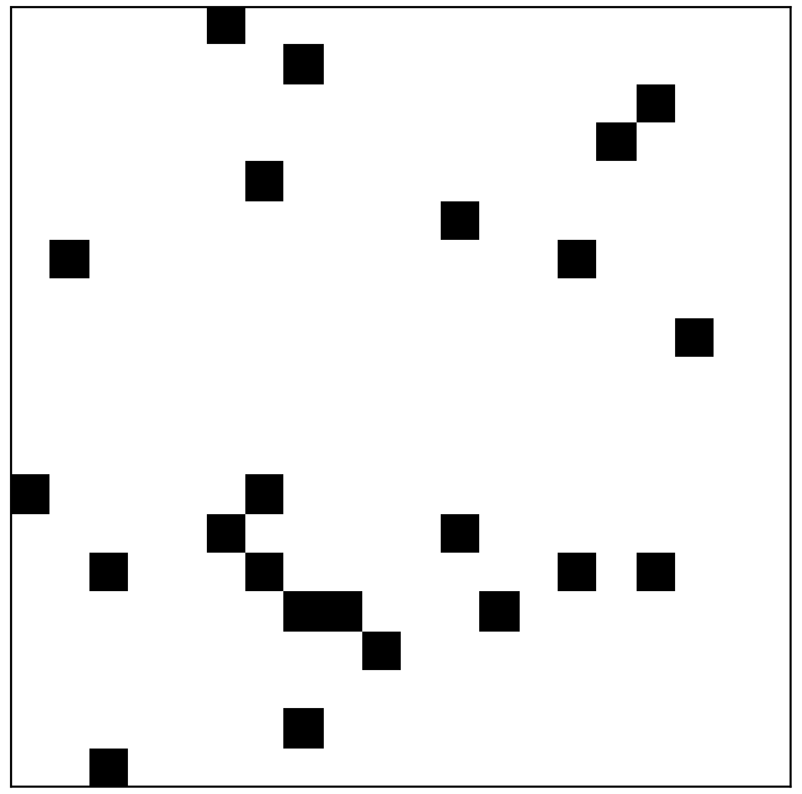
where $\pi = (\pi_1, \ldots, \pi_n)$ is a permutation of $(1, 2, \ldots, n)$

- **interpretation**: $Ax = (x_{\pi_1}, \ldots, x_{\pi_n})$
- Satisfies $A^{-1} = A^T$, hence 0 flops

Sparse matrices

Most **real-world problems** are sparse

A matrix A is **sparse** if the majority of its elements is 0



typically $< 15\%$ nonzeros

Efficient representations

- Triplet format: (i, j, x_{ij})
- Compressed Sparse Column format: (i, x_{ij}) and p_j
- Compressed Sparse Row format: (j, x_{ij}) and p_i

The factor-solve method for solving $Ax = b$

Direct method

1. **Factor** A as a product of simple matrices:

$$A = A_1 A_2 \cdots A_k,$$

   (A_i diagonal, upper/lower triangular, etc)

2. **Compute** $x = A^{-1}b = A_k^{-1} \cdots A_1^{-1} b$ by solving k “easy” equations

$$A_1 x_1 = b_1, \quad A_2 x_2 = x_1, \quad \ldots, \quad A_k x = x_{k-1},$$

   (cost of factorization usually dominates cost of solve)

Multiple righthand sides $Ax = b_1,\ Ax = b_2,\ \ldots,\ Ax = b_m$

cost: one factorization + m solves

(Sparse) LU factorization

Every nonsingular matrix A can be factored as

$$A = P_r L U P_c \quad \longrightarrow \quad P_r^T A P_c^T = LU$$

P_r, P_c permutation, L lower triangular, U upper triangular

Permutations

- Reorder rows P_r and columns P_c of A to (heuristically) get **sparser** L, U
- P_r, P_c depend on sparsity pattern and values of A

Cost

- If A dense, typically $O(n^3)$ but usually much less
- It depends on the number of nonzeros in A, sparsity pattern, etc.

(Sparse) LU solution

$$Ax = b, \quad \Rightarrow \quad P_r L U P_c x = b$$

Iterations

1. *Permutation*: Solve $P_r z_1 = b$ (0 flops)
2. *Forward substitution*: Solve $L z_2 = z_1$ ($O(n^2)$ flops)
3. *Backward substitution*: Solve $U x = z_2$ ($O(n^2)$ flops)
4. *Permutation*: Solve $P_c x = z_2$ (0 flops)

Cost

Factor + Solve $\sim O(n^3)$

Just solve (prefactored) $\sim O(n^2)$

(Sparse) Cholesky factorization

Every positive definite matrix A can be factored as

$$A = PLL^T P^T \longrightarrow P^T AP = LL^T$$

P permutation, L lower triangular

Permutations

- Reorder rows/cols of A with P to (heuristically) get **sparser** L
- P depends only on sparsity pattern of A (unlike LU factorization)
- If A is dense, we can set $P = I$

Cost

- If A dense, typically $O(n^3)$ but usually much less
- It depends on the number of nonzeros in A, sparsity pattern, etc.
- Typically 50% faster than LU (need to find only one matrix)

(Sparse) Cholesky solution

$$Ax = b, \quad \Rightarrow \quad PLL^T P^T x = b$$

Iterations

1. *Permutation*: Solve $Pz_1 = b$ (0 flops)
2. *Forward substitution*: Solve $Lz_2 = z_1$ ($O(n^2)$ flops)
3. *Backward substitution*: Solve $L^T x = z_2$ ($O(n^2)$ flops)
4. *Permutation*: Solve $P^T x = z_2$ (0 flops)

Cost

Factor + Solve $\sim O(n^3)$

Just solve (prefactored) $\sim O(n^2)$

“Realistic” simplex implementation

Computational bottlenecks in the simplex method

Solving linear systems

1. Compute the reduced costs $\bar{c}_j = c_j - c_B^T B^{-1} A_j$ for $j \in N$
4. Compute search direction components $d_B = -B^{-1} A_j$

Equivalent forms

1. Solve $p^T = c_B^T B^{-1} \Rightarrow B^T p = c_B$. Then $\bar{c}_j = c_j - p^T A_j$.
4. Solve $B d_B = -A_j$

Same matrix to factor

$$B^T p = c_B, \quad B d_B = -A_j$$

B not symmetric positive definite $\Rightarrow$ use **LU factorization** $B = LU$
(we here ignore P_r, P_c for simplicity)

Basis update

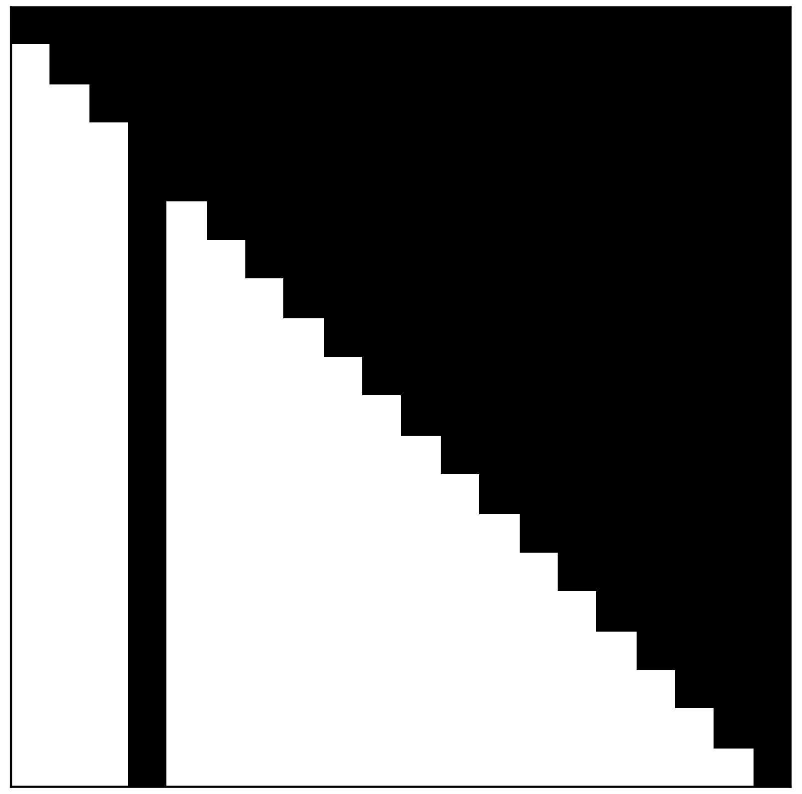
Rank-1 update

$$\bar{B} = B + (A_j - A_i)e_i^T$$

Forrest-Tomlin update $O(n^2)$

- Compute $\bar{B} = LR\bar{U}$ (same L, lower triangular R, upper triangular $\bar{U}$)
- $L^{-1}\bar{B} = U + (L^{-1}A_j - Ue_i)e_i^T = \hat{U}$
- LU factorization of $\hat{U}$ into $R\bar{U}$ via **elimination** (cheap)

$\hat{U}$



Remarks

- Implemented in modern sparse solvers
- Accumulates errors (we need to refactor B from scratch once in a while)
- Many more algorithms: Block-LU, Bartels-Golub-Reid, etc.

Realistic (revised) simplex method

Initialization

1. Given basic feasible solution x
2. Factor basis matrix $B = \begin{bmatrix} A_{B(1)} & \dots, A_{B(m)} \end{bmatrix}$

Iterations

1. Solve $B^T p = c_B, \quad$ ($O(m^2)$)
2. Compute the reduced costs. $\bar{c} = c - A^T p$
3. If $\bar{c} \geq 0$, x **optimal**. **break**
4. Choose j such that $\bar{c}_j < 0$

Realistic (revised) simplex method

Iterations (continued)

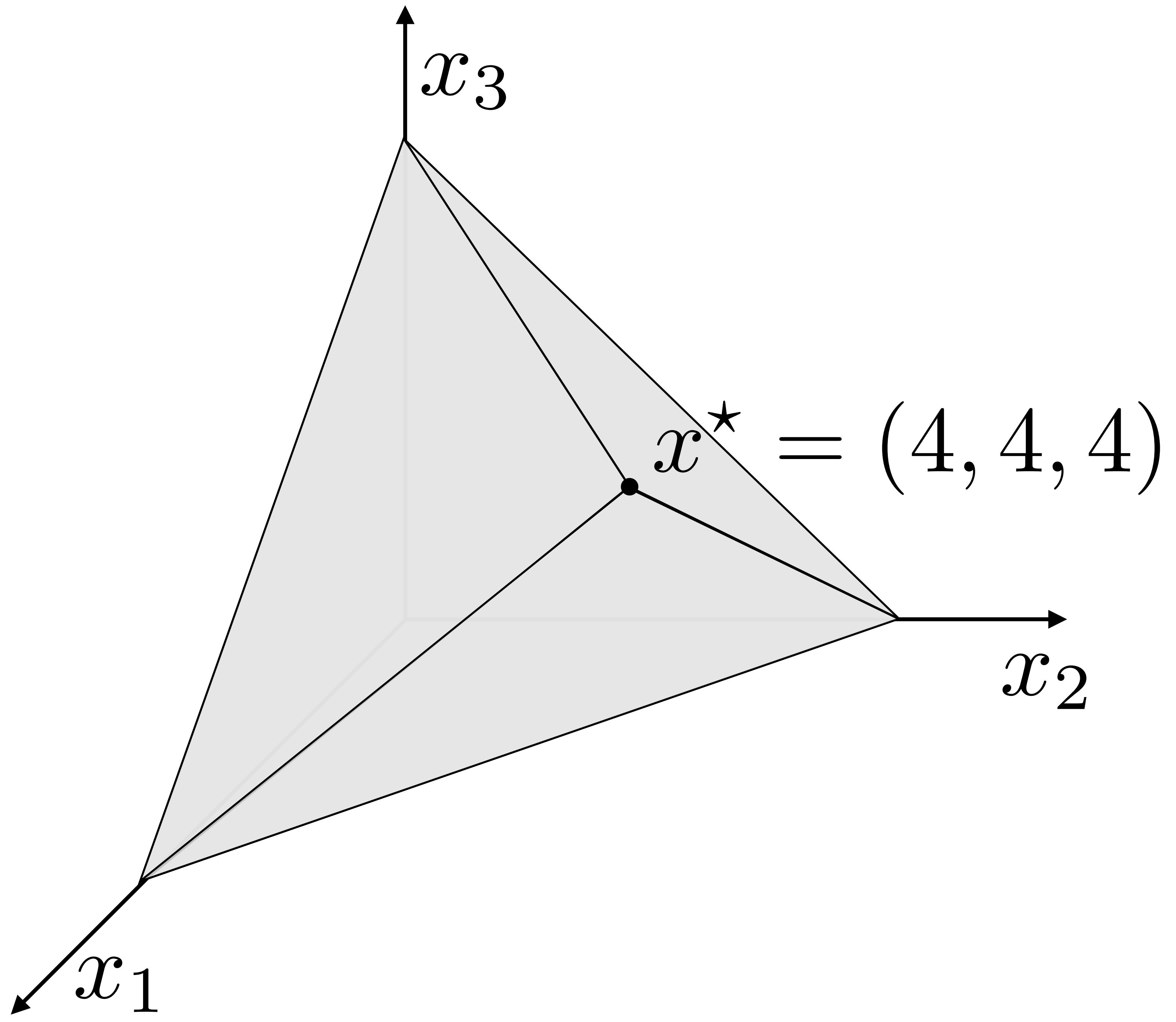
5. Compute search direction. $d_j = 1$ and solve $Bd_B = -A_j$ $\quad (O(m^2))$
6. If $d_B \geq 0$, the problem is **unbounded** and the optimal value is $-\infty$. **break**
7. Compute step length $\theta^\star = \min_{\{i \in B | d_i < 0\}} \left(-\frac{x_i}{d_i}\right)$ and pick i exiting the basis
8. Compute new point $y = x + \theta^\star d$
9. Get new basis $\bar{B} = B + (A_j - A_i)e_i^T$ and perform rank-1 factor update. $\quad (O(m^2))$

Per-iteration cost $O(m^2)$

Example

Example

$$\begin{array}{ll} \text{minimize} & -10x_1 - 12x_2 - 12x_3 \\ \text{subject to} & x_1 + 2x_2 + 2x_3 \leq 20 \\ & 2x_1 + x_2 + x_3 \leq 20 \\ & 2x_1 + 2x_2 + x_3 \leq 20 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$



Standard form

minimize $\quad -10x_1 - 12x_2 - 12x_3$

subject to
$$\begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix}$$

$$x \geq 0$$

Example

Start

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

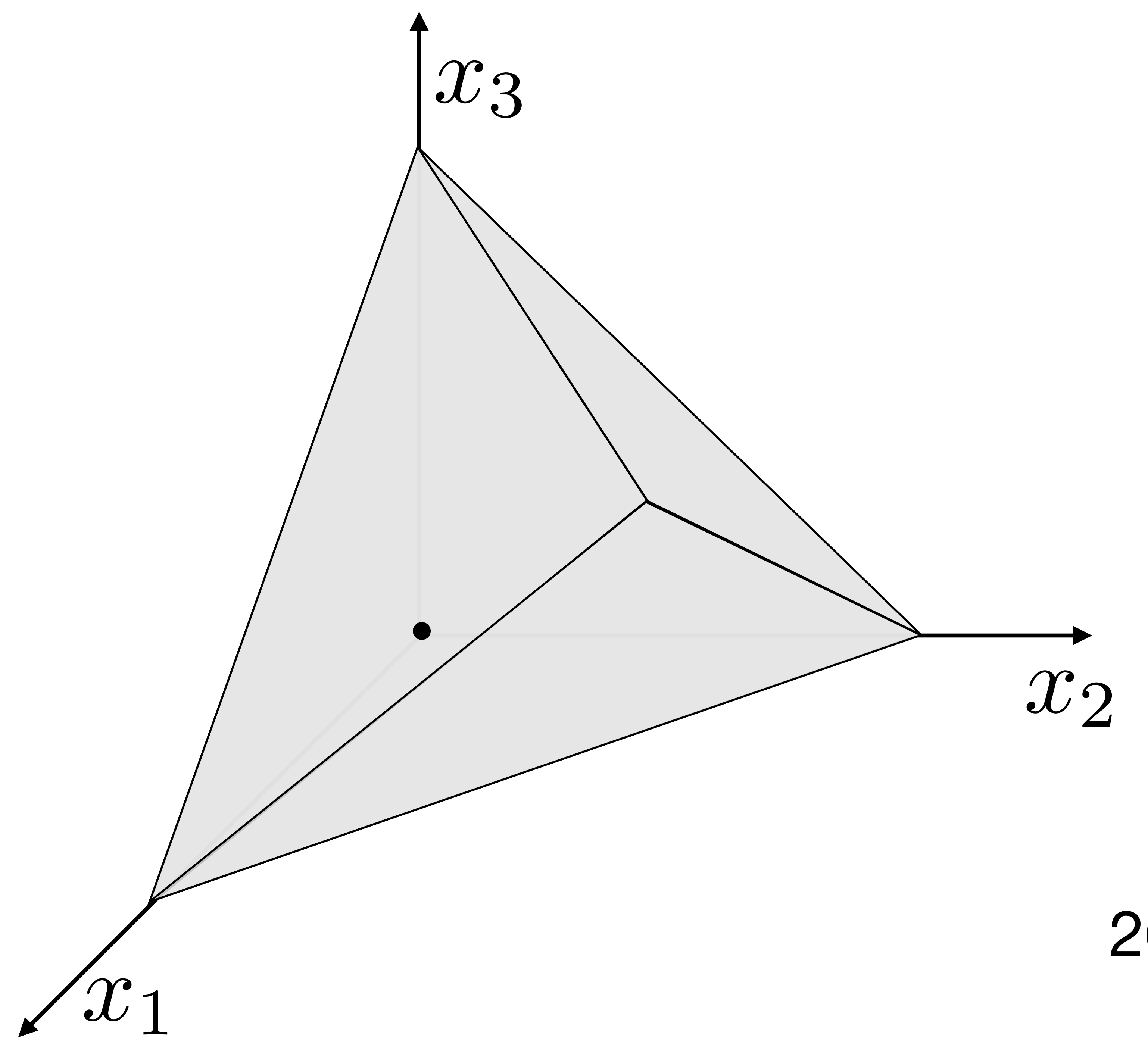
$$c = (-10, -12, -12, 0, 0, 0)$$

$$A = \begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$b = (20, 20, 20)$$

Initialize

$$x = (0, 0, 0, 20, 20, 20) \qquad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Example

Iteration 1

Current point

$x = (0, 0, 0, 20, 20, 20)$
$c^T x = 0$

Basis: $\{4, 5, 6\}$

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$c = (-10, -12, -12, 0, 0, 0)$

$$A = \begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$b = (20, 20, 20)$

Reduced costs $\bar{c} = c$

Solve $B^T p = c_B \quad \Rightarrow \quad p = c_B = 0$

$\bar{c} = c - A^T p = c$

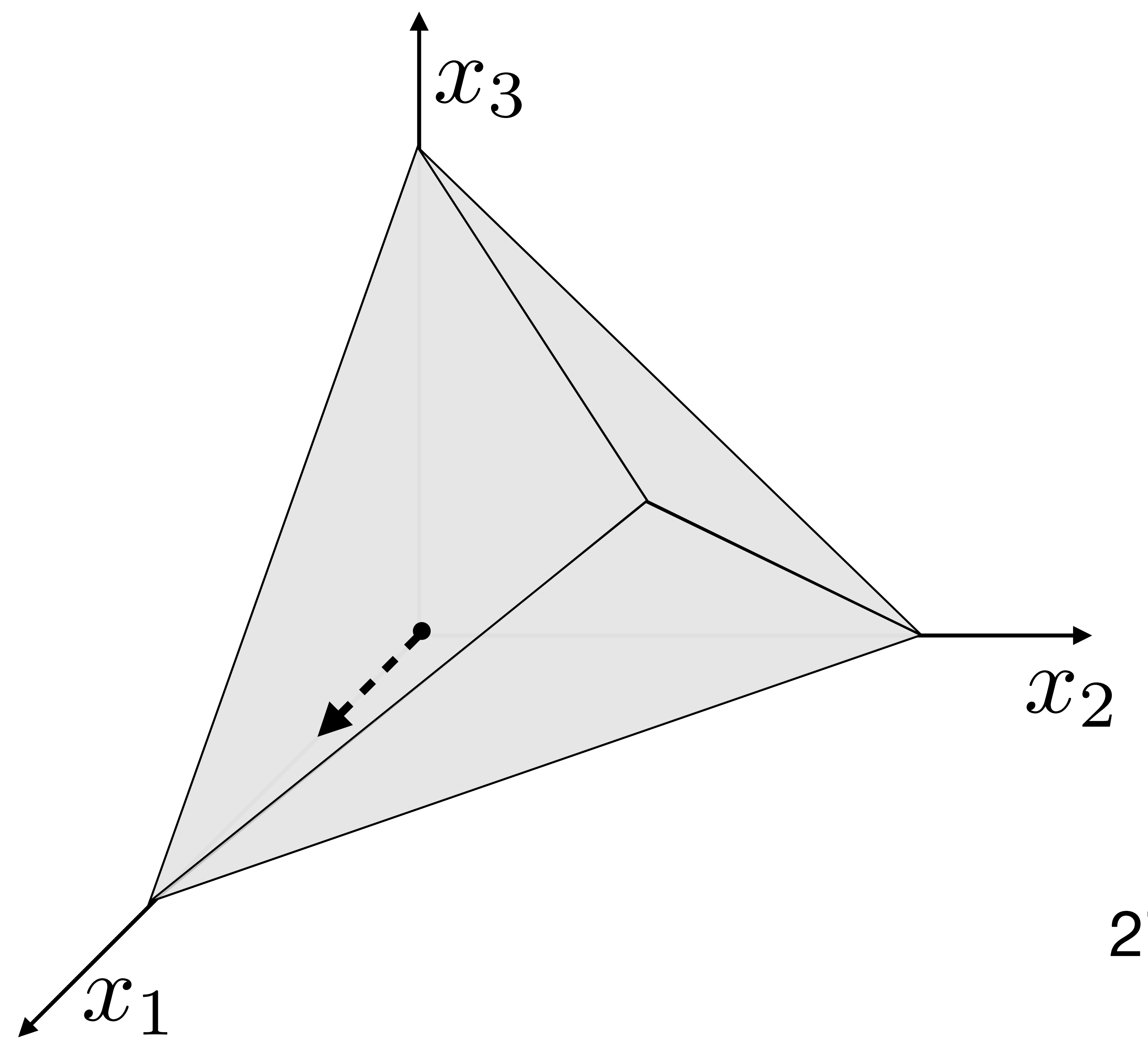
Direction $d = (1, 0, 0, -1, -2, -2), \quad j = 1$

Solve $B d_B = -A_j \quad \Rightarrow \quad d_B = (-1, -2, -2)$

Step $\theta^\star = 10, \quad i = 5$

$$\theta^\star = \min_{\{i | d_i < 0\}} (-x_i / d_i) = \min\{20, 10, 10\}$$

New $x \leftarrow x + \theta^\star d = (10, 0, 0, 10, 0, 0)$



Example

Iteration 2

Current point

$x = (10, 0, 0, 10, 0, 0)$
$c^T x = -100$

Basis: $\{4, 1, 6\}$

$$B = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

$c = (-10, -12, -12, 0, 0, 0)$

$$A = \begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$b = (20, 20, 20)$

Reduced costs $\bar{c} = (0, -7, -2, 0, 5, 0)$

Solve $B^T p = c_B \quad \Rightarrow \quad p = (0, -5, 0)$

$\bar{c} = c - A^T p = (0, -7, -2, 0, 5, 0)$

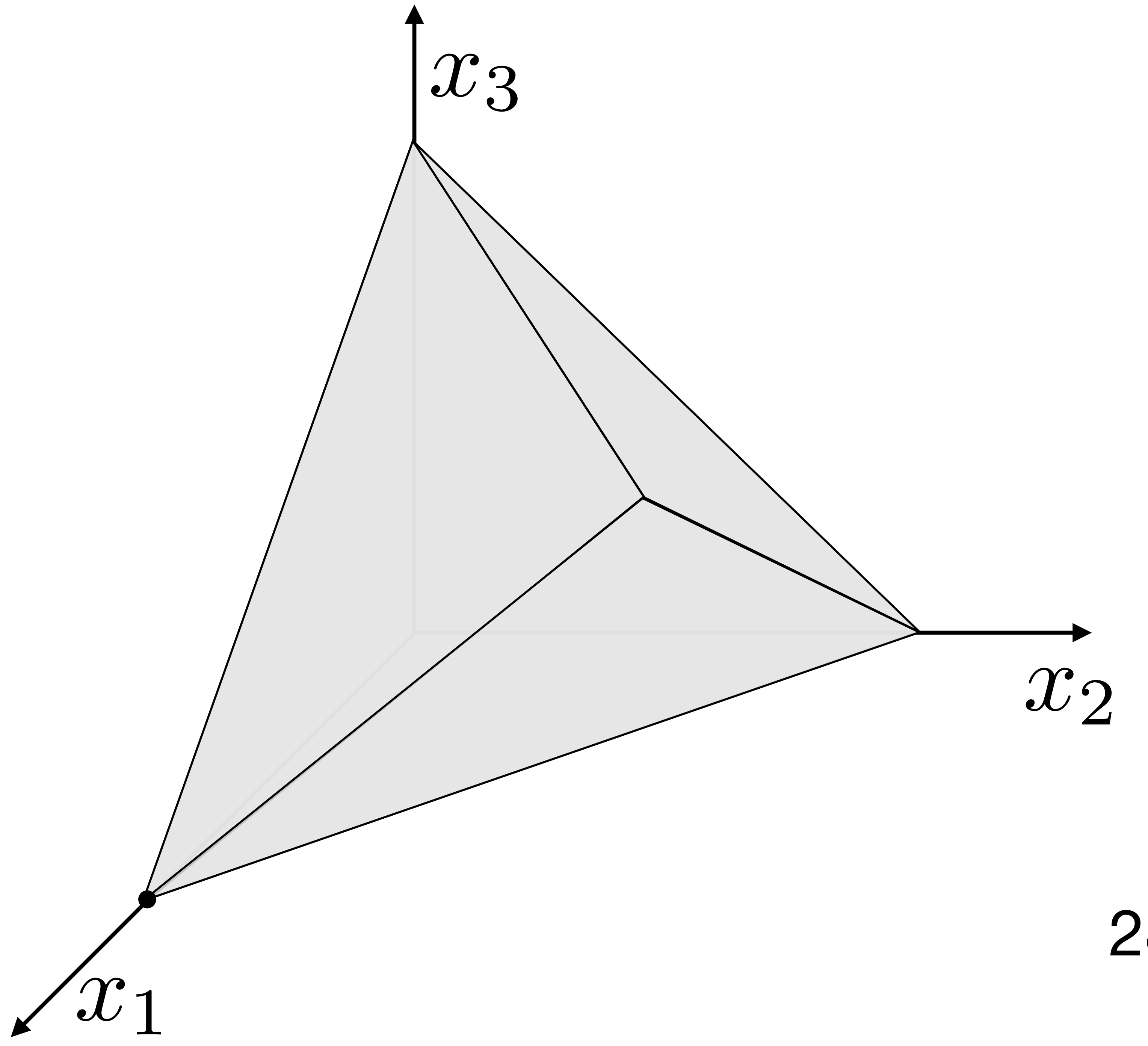
Direction $d = (-0.5, 1, 0, -1.5, 0, -1), \quad j = 2$

Solve $B d_B = -A_j \quad \Rightarrow \quad d_B = (-1.5, -0.5, -1)$

Step $\theta^\star = 0, \quad i = 6$

$$\theta^\star = \min_{\{i | d_i < 0\}} (-x_i / d_i) = \min\{6.66, 20, 0\}$$

New $x \leftarrow x + \theta^\star d = (10, 0, 0, 10, 0, 0)$



Example
Iteration 3

Current point

$x = (10, 0, 0, 10, 0, 0)$
$c^T x = -100$

Basis: $\{4, 1, 2\}$

$$B = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 2 & 1 \\ 0 & 2 & 2 \end{bmatrix}$$

$c = (-10, -12, -12, 0, 0, 0)$

$$A = \begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$b = (20, 20, 20)$

Reduced costs $\bar{c} = (0, 0, -9, 0, -2, 7)$

Solve $B^T p = c_B \quad \Rightarrow \quad p = (0, 2, -7)$

$\bar{c} = c - A^T p = (0, 0, -9, 0, -2, 7)$

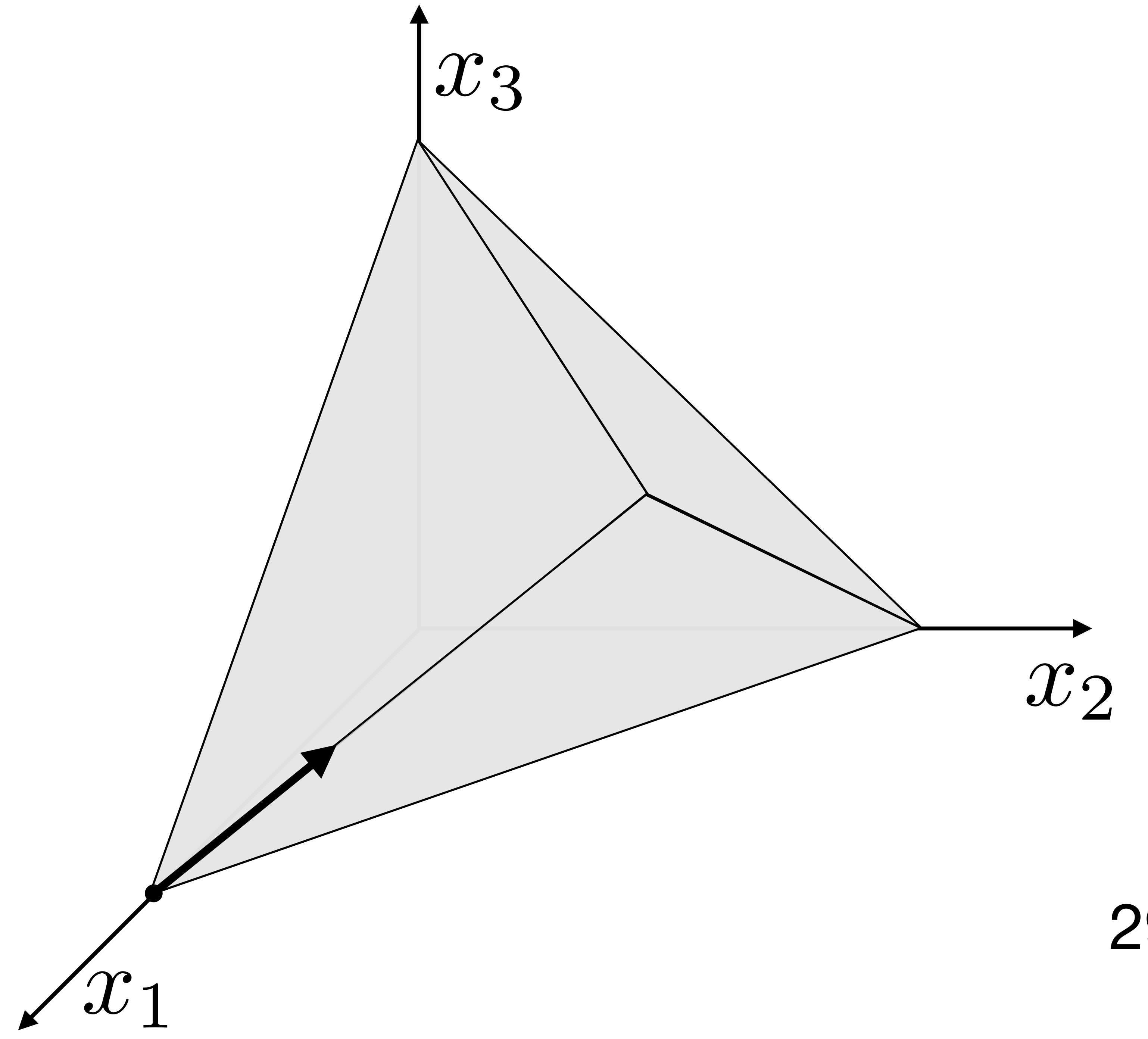
Direction $d = (-1.5, 1, 1, -2.5, 0, 0), \quad j = 3$

Solve $B d_B = -A_j \quad \Rightarrow \quad d_B = (-2.5, -1.5, 1)$

Step $\theta^\star = 4, \quad i = 4$

$$\theta^\star = \min_{\{i | d_i < 0\}} (-x_i / d_i) = \min\{4, 6.67\}$$

New $x \leftarrow x + \theta^\star d = (4, 4, 4, 0, 0, 0)$



Example

Iteration 4

Current point

$x = (4, 4, 4, 0, 0, 0)$

$c^T x = -136$

Basis: $\{3, 1, 2\}$

$$B = \begin{bmatrix} 2 & 1 & 2 \\ 2 & 2 & 1 \\ 1 & 2 & 2 \end{bmatrix}$$

$c = (-10, -12, -12, 0, 0, 0)$

$$A = \begin{bmatrix} 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 1 & 0 \\ 2 & 2 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$b = (20, 20, 20)$

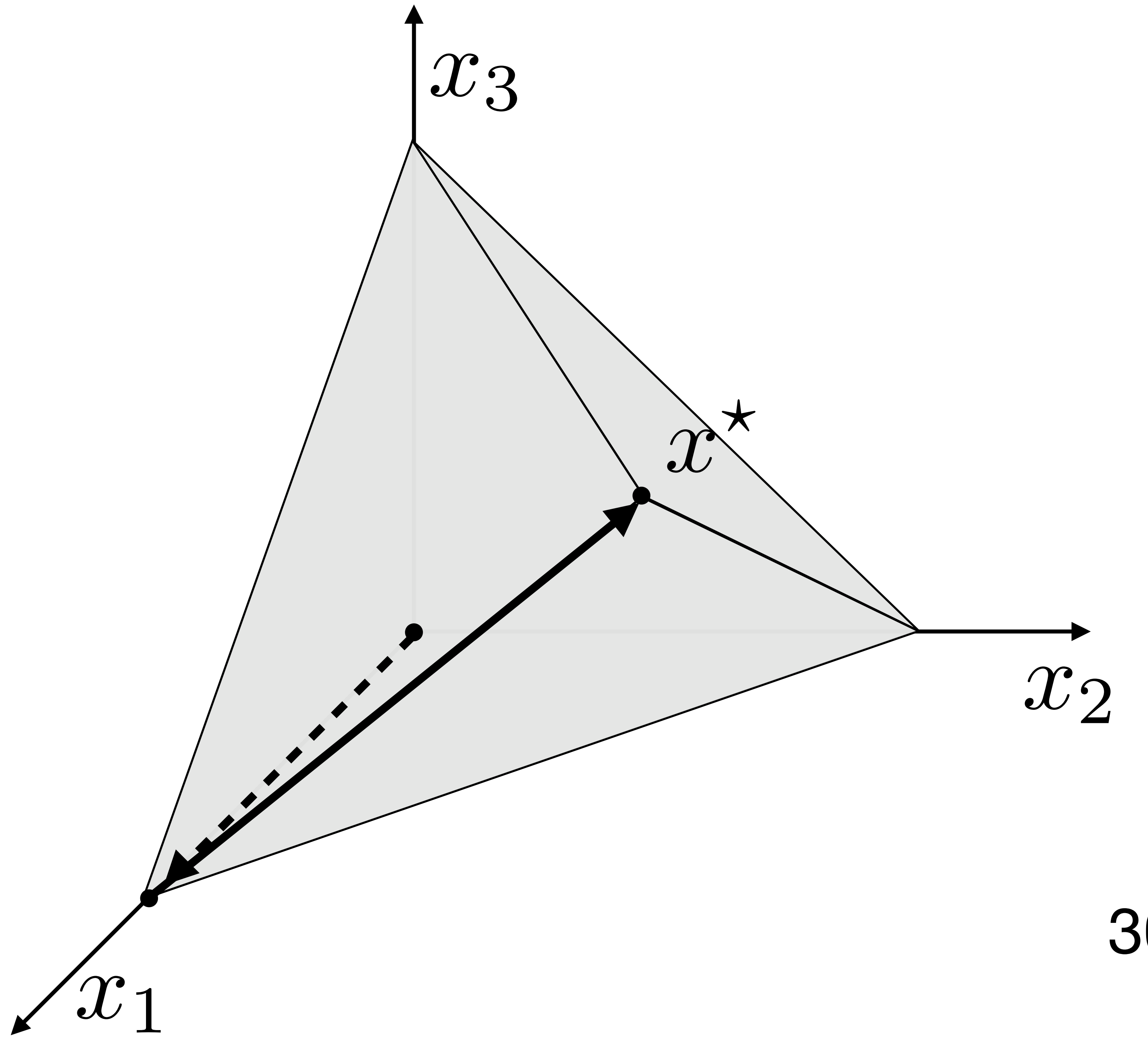
Reduced costs $\bar{c} = (0, 0, 0, 3.6, 1.6, 1.6)$

Solve $B^T p = c_B \quad \Rightarrow \quad p = (-3.6, -1.6, -1.6)$

$\bar{c} = c - A^T p = (0, 0, 0, 3.6, 1.6, 1.6)$

Optimal

$\bar{c} \geq 0 \longrightarrow x^\star = (4, 4, 4, 0, 0, 0)$



Simplex tableau implementation

Can we solve LPs by hand?

Minus cost →	$-c_B^T x_B$	$\bar{c}_1 \quad \dots \quad \bar{c}_n$	← Reduced costs
Basic variables →	$x_B(1)$ $\vdots$ $x_B(1)$	$B^{-1}A_1 \quad \dots \quad B^{-1}A_n$	

$$
\begin{array}{c|ccc}
-c_B^T x_B & \bar{c}_1 & \dots & \bar{c}_n \\
\hline
x_B(1) & | & & | \\
\vdots & B^{-1}A_1 & \dots & B^{-1}A_n \\
x_B(1) & | & & |
\end{array}
$$

People did it **before computers were invented**!

Nobody does it anymore…

Empirical complexity

Example with real solver

GLPK (open-source)

Code

```
import numpy as np
import cvxpy as cp

c = np.array([-10, -12, -12])
A = np.array([[1, 2, 2],
              [2, 1, 2],
              [2, 2, 1]])
b = np.array([20, 20, 20])
n = len(c)

x = cp.Variable(n)
problem = cp.Problem(cp.Minimize(c @ x),
                     [A @ x <= b, x >= 0])
problem.solve(solver=cp.GLPK, verbose=True)
```

Output

```
GLPK Simplex Optimizer, v4.65
6 rows, 3 columns, 12 non-zeros
*     0: obj =   0.000000000e+00 inf =   0.000e+00 (3)
*     3: obj =  -1.360000000e+02 inf =   0.000e+00 (0)
OPTIMAL LP SOLUTION FOUND
```

Average simplex complexity

Random LPs

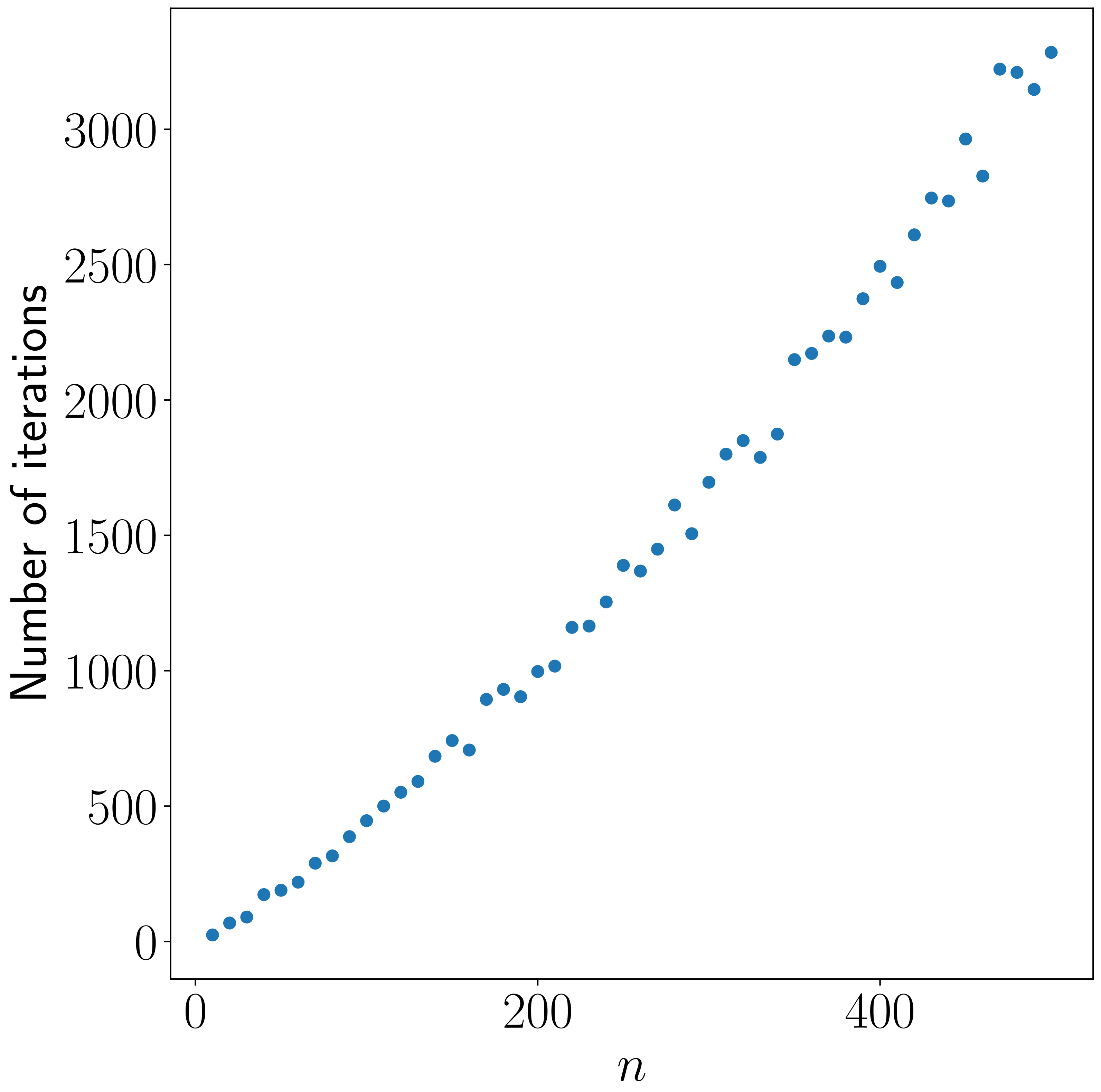
minimize $c^T x$

subject to $Ax \leq b$

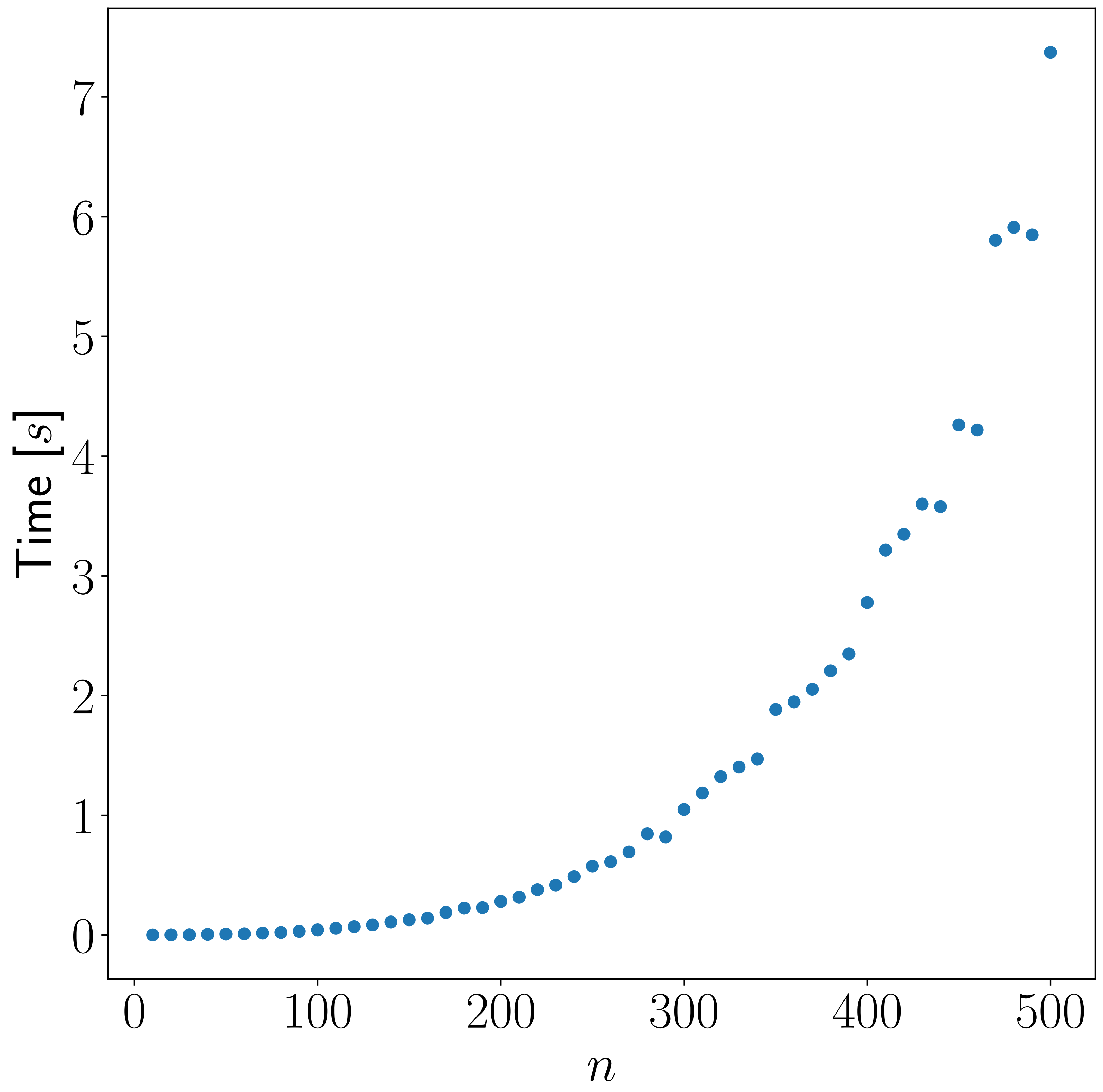
n variables

$3n$ constraints

Iterations: $O(n)$



Time: $O(nn^2) = O(n^3)$



Numerical linear algebra and simplex implementation

Today, we learned to:

- **Identify** the pros and cons of different methods to solve a linear system
- **Derive** the computational complexity of the factor-solve method
- **Implement** a “realistic” version of the simplex method
- **Empirically** analyze the average complexity of the simplex method

Next lecture

- Linear optimization duality