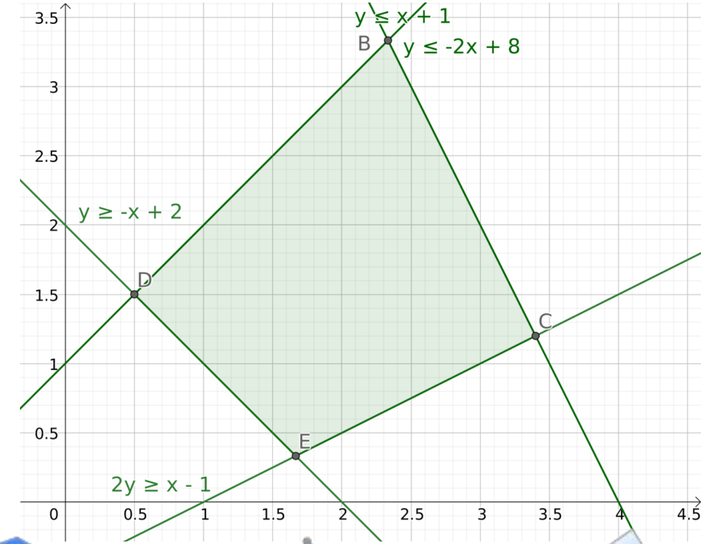
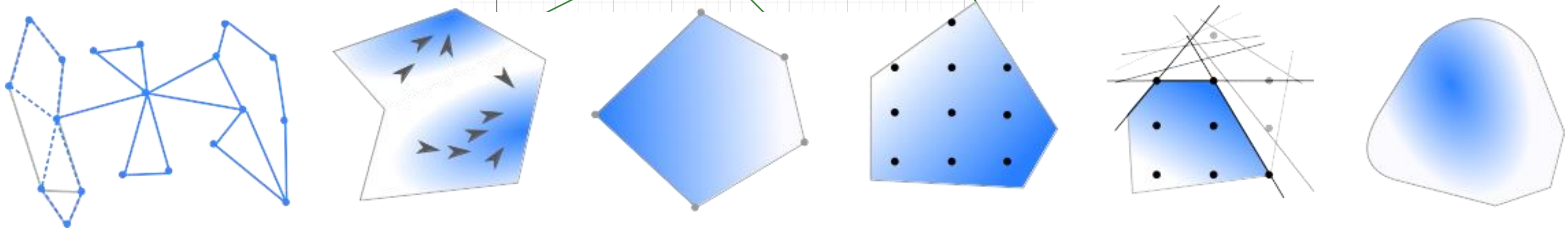
Linear Programming 4

COMP4691 / 8691





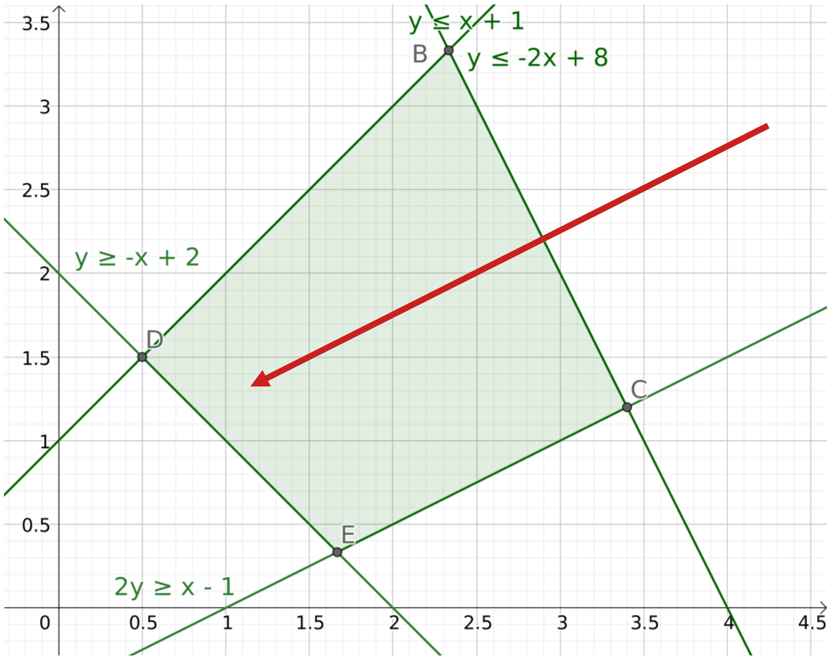
LP Topic Outline

- LP Introduction
- Modelling and solving
- Feasible region and convexity
- **Simplex algorithm**
    - Alternative Simplex algorithm
    - Degeneracy
    - Complexity
    - Bootstrapping
    - BFS forms
- Relaxations and approximations
- The dual of a linear program

Solving an LP

Simplex algorithm: hop from vertex to neighbouring vertex so long as there is an improvement in objective
1947 George Dantzig (not polynomial time but performance typically good)

Interior point algorithm: traverse the interior of the feasible region following gradient of objective while avoiding the boundaries
1947 John von Neumann (not polynomial time, and typically worse than simplex)



Will explore recent (polynomial time) version in convex optimisation topic

Simplex Algorithm Forms

Many different mathematically equivalent formulations. Most work with LPs in following form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

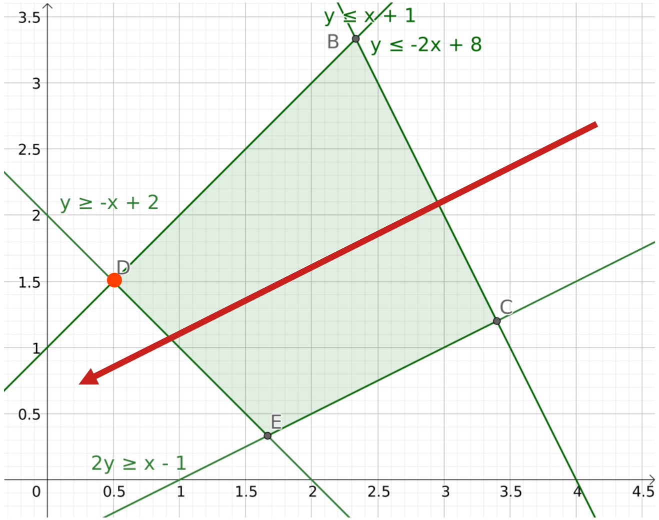
- **Standard** Simplex Algorithm: tableau manipulation
- **Revised** Simplex Algorithm: matrix manipulation

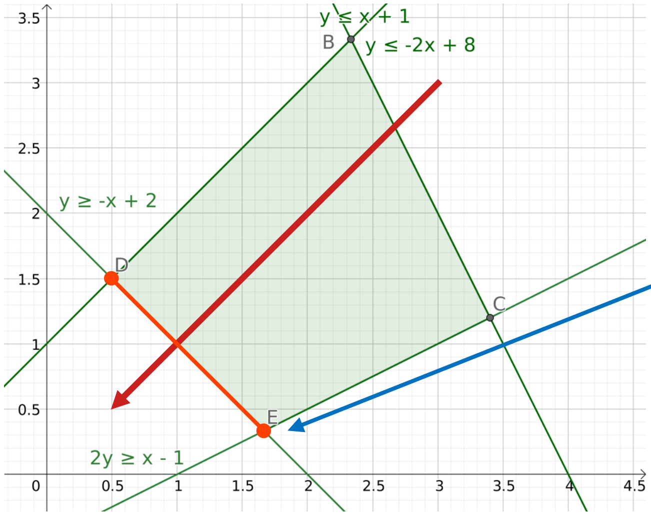
We'll initially break convention in order to give a better geometric intuition, by focusing on solving LPs in this form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

- **"Alternative"** Simplex Algorithm: matrix manipulation

Simplex Algorithm





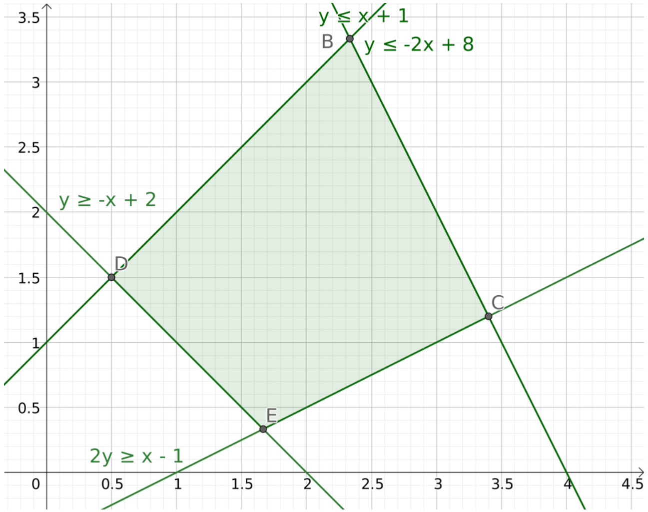
Infinite number of optimal solutions: still at least one of them is on a vertex!

LPs: if the **problem** is feasible and bounded, then there exists at least one extreme point (vertex of polyhedron) that is optimal.

<u>Simplex insight</u>: if we want **an** optimal solution, let’s “just” search the vertices!

Finding Vertices

How can we (algorithmically) find vertices?



$$\min_{x \in \mathbb{R}^n} \; c^\intercal x$$

$$\text{s.t. } Ax \leq b$$

A vertex in $\mathbb{R}^n$ is defined by the intersection of **n** linearly independent hyperplanes (associated with n equality or active inequality constraints)

Will all constraint combinations form vertices of the feasible region?

An Approach to Finding Vertices

$$\min_{x \in \mathbb{R}^n} \quad c^\intercal x$$

$$\text{s.t. } Ax \leq b$$

1. Pick n linearly independent constraints

2. Find their intercept by solving the n x n system:

$$\bar{A}x^* = \bar{b}$$

$\bar{A}, \bar{b}$ have rows for non-active constraints removed

3. Check if candidate x^* is a vertex of the feasible region:

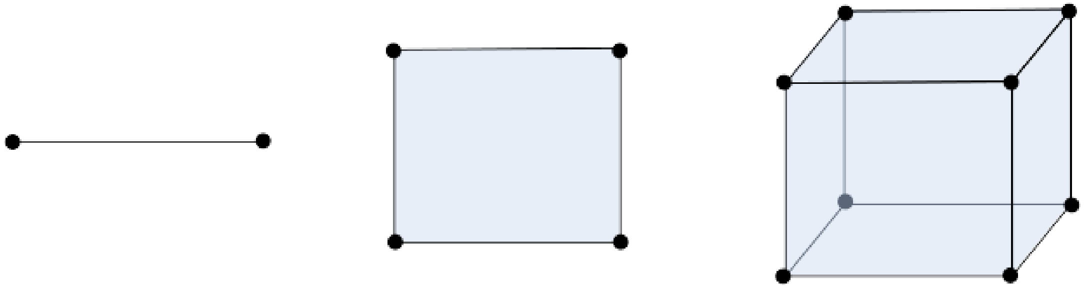
$$Ax^* \leq b$$

How many Vertices are there?

Can we just enumerate them?

This approach requires checking **m choose n** combinations.

Even if we could limit ourselves to true vertices, their number can still **grow exponentially** in the problem size:



Hypercube

n	1	2	3	n
m	2	4	6	2n
vertices	2	4	8	2^n

Brute forcing it will lead to sorrow!

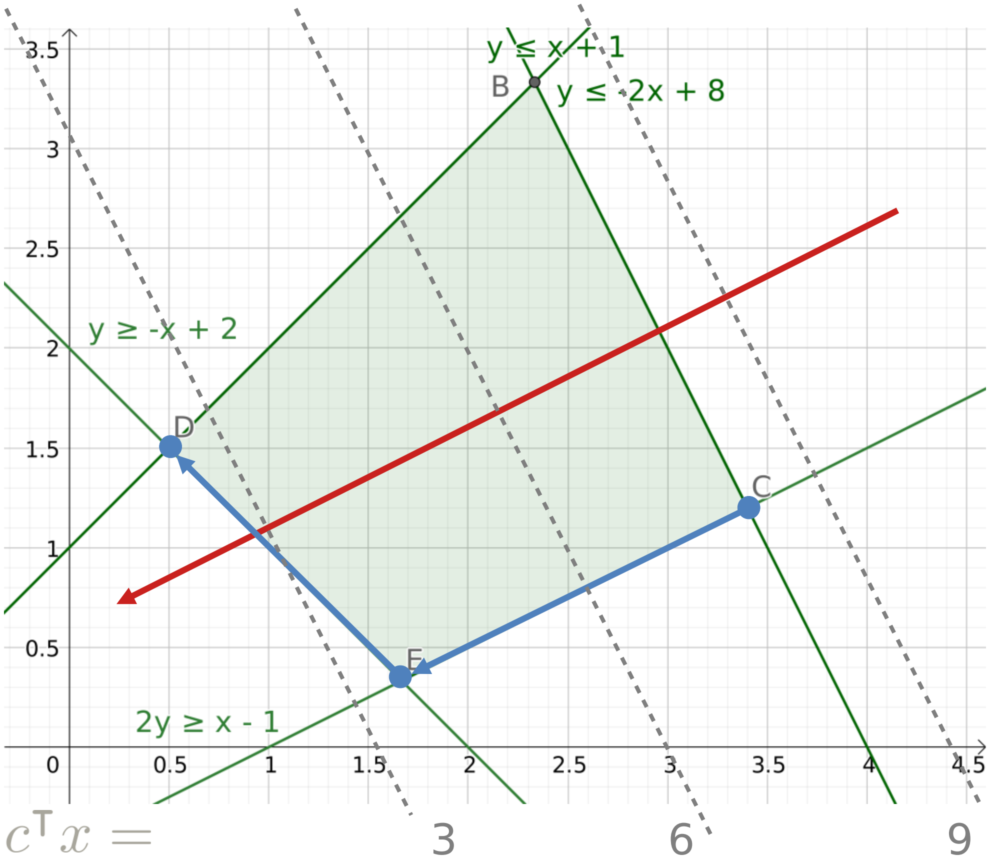
Simplex Algorithm

The **Simplex algorithm** is smarter than this. It moves from **vertex to vertex**, picking a new neighbour (sharing all but one constraint) each time that improves the objective function.

Assuming there are no "degenerate" vertices (for later):

- **If there are no neighbouring vertices that improve the objective, then our current vertex is optimal**
- If we only move to vertices that improve the objective, we will converge to an optimal solution in a finite number of steps.

Simplex Algorithm



“Alternative” Simplex Overview

$$\min_{x \in \mathbb{R}^n} \; c^\intercal x$$
$$\text{s.t. } Ax \le b$$

$A \in \mathbb{R}^{m \times n}$

$A_{i,j} \in \mathbb{R}$ element

$A_i \in \mathbb{R}^{1 \times n}$ i-th row

$A_{:,j} \in \mathbb{R}^{m \times 1}$ j-th column

At the k-th iteration, **a vertex is represented by $\mathbf{S_k}$**, the **active constraint set**:

$$S_k \subseteq \{1, \ldots, m\} \qquad |S_k| = n$$

Find a constraint to leave S_k and a new one to enter to get S_{k+1}, i.e., the vertex at the next iteration. This is called **pivoting**.

But how do we do this so:

- the objective improves?
- the new active constraint set is a vertex of the feasible region?

Vertex Position

Active constraint set for each iteration k: $S_k \subseteq \{1, \ldots, m\}$

$|S_k| = n$

Vertex position: $Ax \leq b \longrightarrow \bar{A}x_k = \bar{b}$

$n \times n$ system to solve

Take rows of current active constraint set

Zero slack when active so equality

Vertex Position from 3-D Example

$$A = \begin{bmatrix} -0.80804753 & -0.58025801 & 0.10178351 \\ -0.50084776 & -0.71380377 & 0.48952585 \\ -0.08048842 & -0.09915476 & 0.99181145 \\ -0.09627039 & 0.0453119 & 0.99432325 \\ -0.17546873 & 0.20627031 & 0.96263349 \\ -0.75252348 & 0.64080679 & -0.1519043 \\ -0.95336509 & 0.1294336 & -0.27265716 \\ -0.26776844 & -0.72117537 & -0.63891017 \\ -0.34649301 & -0.61552572 & -0.70786339 \\ -0.49174464 & -0.7193774 & 0.49059495 \\ -0.19321021 & -0.77477372 & -0.60199296 \\ 0.04926607 & -0.39601576 & 0.91692108 \\ 0.98859793 & 0.01616362 & 0.14970967 \\ 0.77713436 & 0.18456472 & 0.60166258 \\ -0.01748661 & 0.08323177 & 0.99637675 \\ 0.07115179 & -0.40672514 & 0.91077548 \\ 0.88839614 & -0.24955757 & 0.38532227 \\ 0.99100763 & -0.03015041 & 0.13036372 \\ -0.1961358 & 0.78581196 & 0.58654082 \\ -0.19422084 & 0.83695108 & 0.51165545 \\ -0.22796361 & 0.91862208 & 0.32274759 \\ -0.62808508 & 0.74864495 & -0.21222581 \\ -0.21458624 & 0.75690502 & -0.6172905 \\ -0.11244072 & 0.65154088 & -0.75023437 \\ -0.06446069 & -0.79907358 & -0.59776777 \\ 0.55077285 & 0.32070059 & -0.77058434 \\ 0.55246085 & 0.3186858 & -0.77021194 \\ 0.57675749 & -0.43490228 & -0.6915279 \\ 0.41864392 & -0.72648567 & 0.54493666 \\ 0.94824499 & 0.31748062 & -0.00612433 \\ 0.97529268 & -0.07944221 & -0.20613866 \\ -0.02390716 & 0.99852639 & -0.04871828 \end{bmatrix}$$

$$b = \begin{bmatrix} -0.1990183 \\ 0.07182385 \\ 0.81146578 \\ 0.88493254 \\ 0.9367934 \\ 0.18363547 \\ -0.17376207 \\ -0.57688478 \\ -0.58442834 \\ -0.55015062 \\ 0.68138392 \\ 1.03655597 \\ 1.37585999 \\ 0.97976262 \\ 0.68881921 \\ 1.01392535 \\ 0.99302257 \\ 1.08198406 \\ 1.06040375 \\ 0.95691505 \\ 0.25421719 \\ 0.24125601 \\ 0.16925482 \\ -0.47680616 \\ 0.44527668 \\ 0.44545848 \\ 0.10937272 \\ 0.611702 \\ 1.16971561 \\ 0.78748474 \\ 0.92011539 \end{bmatrix}$$

$$c = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.0 \end{bmatrix} \qquad S_k = \{15, 16, 28\}$$

$$\bar{A} = \begin{bmatrix} 0.07115179 & -0.40672514 & 0.91077548 \\ 0.88839614 & -0.24955757 & 0.38532227 \\ 0.41864392 & -0.72648567 & 0.54493666 \end{bmatrix}$$

$$\bar{b} = \begin{bmatrix} 0.68881921 \\ 1.01392535 \\ 0.611702 \end{bmatrix}$$

Solve:

$$\bar{A}x_k = \bar{b} \qquad x_k = \begin{bmatrix} 0.86530995 \\ 0.26049233 \\ 0.80502786 \end{bmatrix}$$

Verify a vertex? i.e. $Ax_k \leq b$

For most real, high-dimensional problems this is actually sparse.

Edges Lead to Neighbours

If we move along an edge we will sooner or later reach another vertex (assuming a bounded problem).

Edges (lines) connected to our current vertex are represented by a subset of **n – 1 constraints**:

$$S_k \setminus \{u\}$$

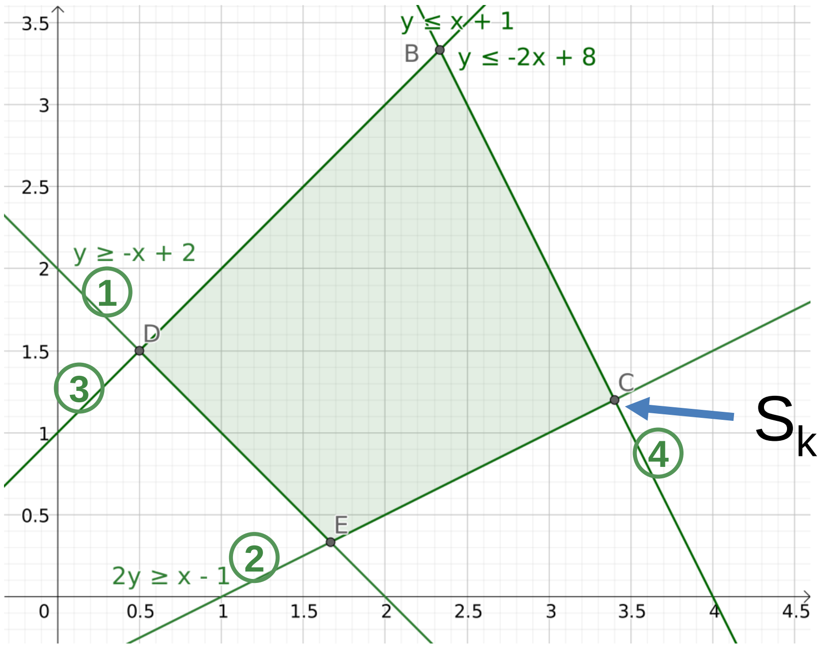
Example: $S_k = \{2,4\}$

1: $y \geq -x + 2$

2: $2y \geq x - 1$

3: $y \leq x + 1$

4: $y \leq -2x + 8$

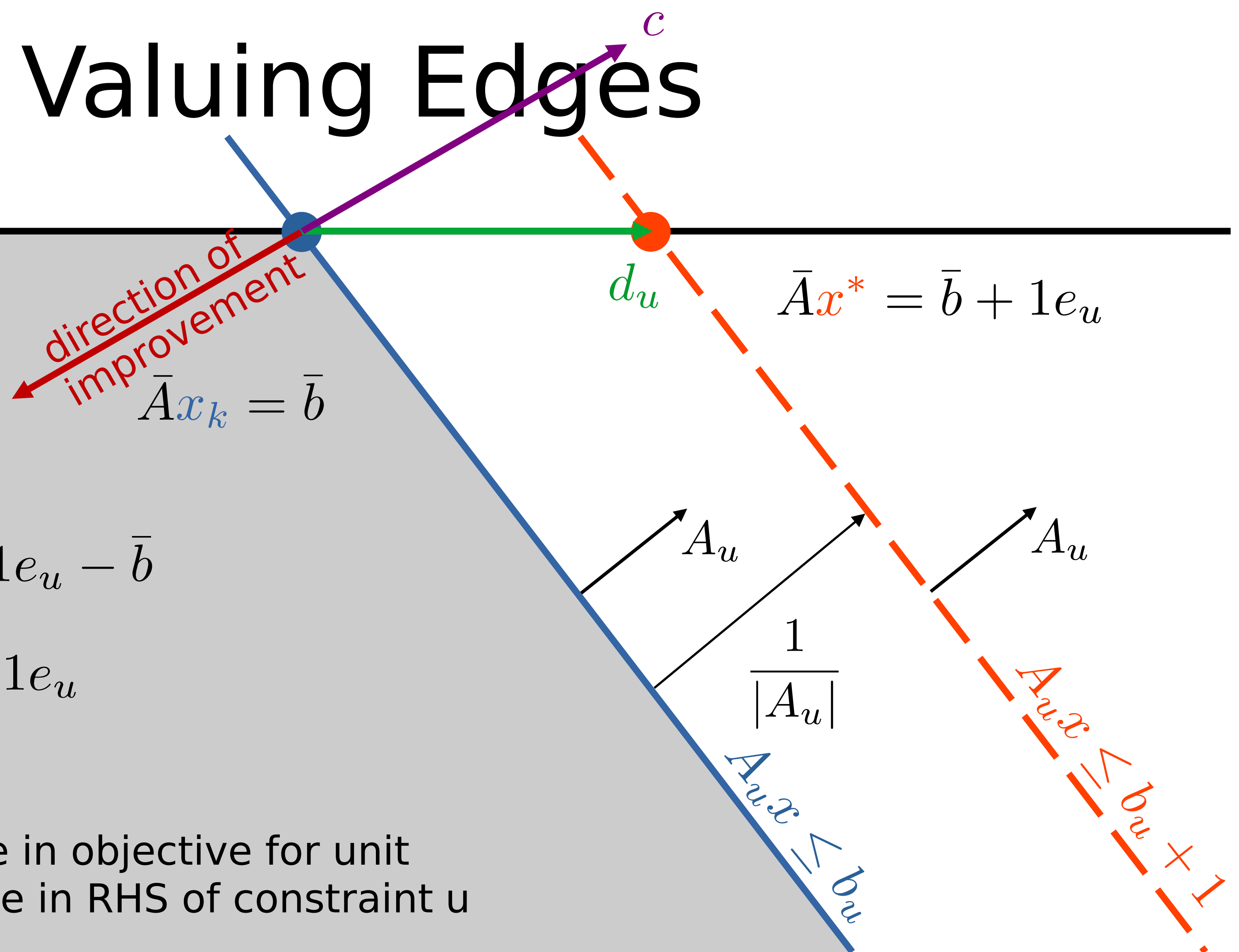


$S_k = \{2,4\}$

Valuing Edges

$$\min_{x \in \mathbb{R}^n} \quad c^\intercal x$$
$$\text{s.t. } Ax \le b$$

Edge

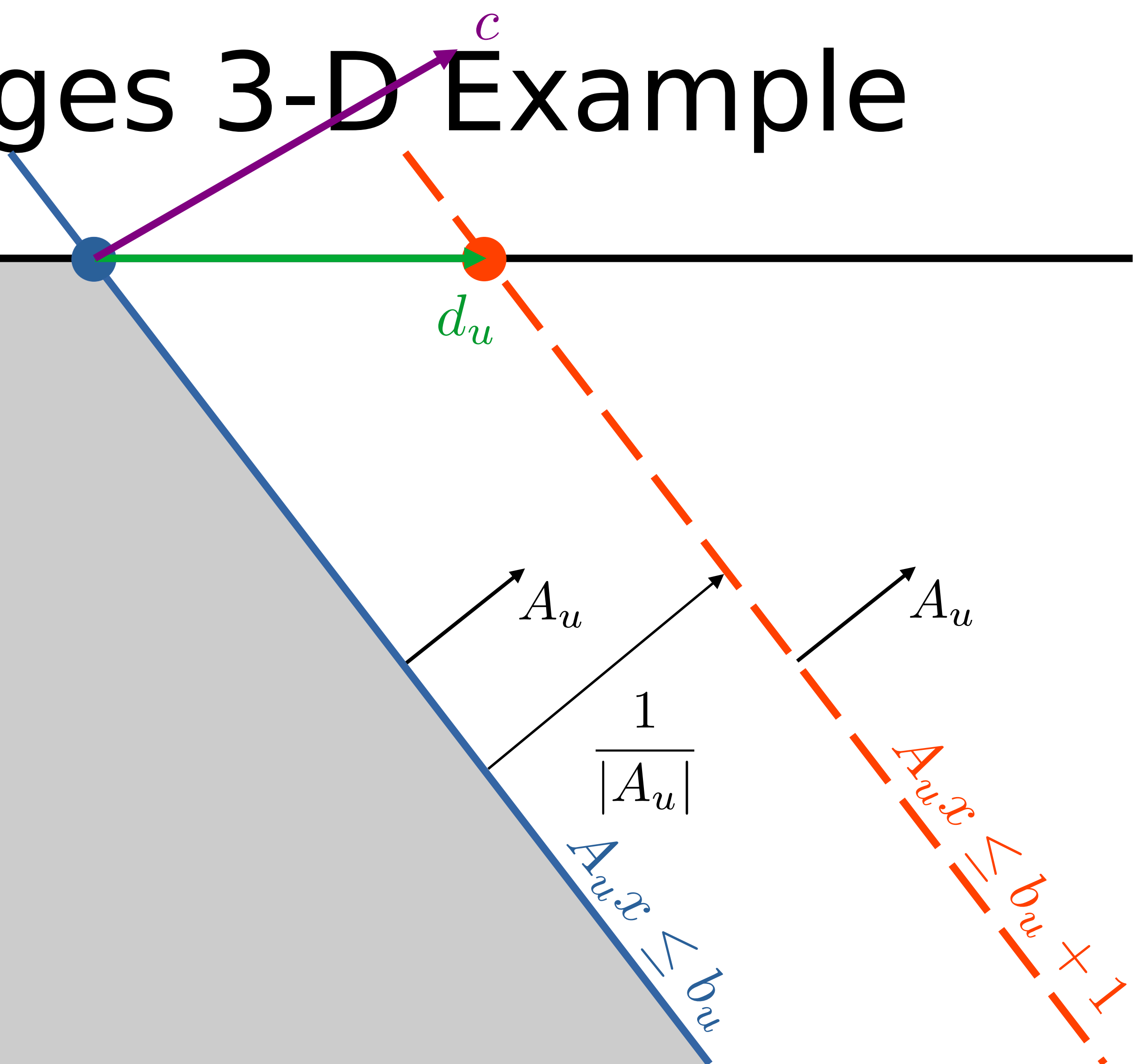


$$\bar{A}x^* - \bar{A}x_k = \bar{b} + 1e_u - \bar{b}$$
$$\implies \bar{A}(x^* - x_k) = 1e_u$$
$$\implies \bar{A}d_u = 1e_u$$

$\mu_u = d_u{}^\intercal c$ Change in objective for unit increase in RHS of constraint u

Valuing Edges 3-D Example

Edge



$$\bar{A} d_u = 1 e_u$$

$$\mu_u = d_u^{\intercal} c$$

$$u = 15: \quad d_u = \begin{bmatrix} -0.40992586 \\ 0.91933208 \\ 1.54053601 \end{bmatrix}$$

$$\mu_u = 0.255$$

$$u = 16: \quad d_u = \begin{bmatrix} 1.25316418 \\ 0.97546654 \\ 0.33771428 \end{bmatrix}$$

$$\mu_u = 1.114$$

Valuing All Edges

Edge

$$\bar{A} d_u = 1 e_u \qquad \bar{A} x_k = \bar{b}$$

$$\mu_u = {d_u}^\intercal c$$

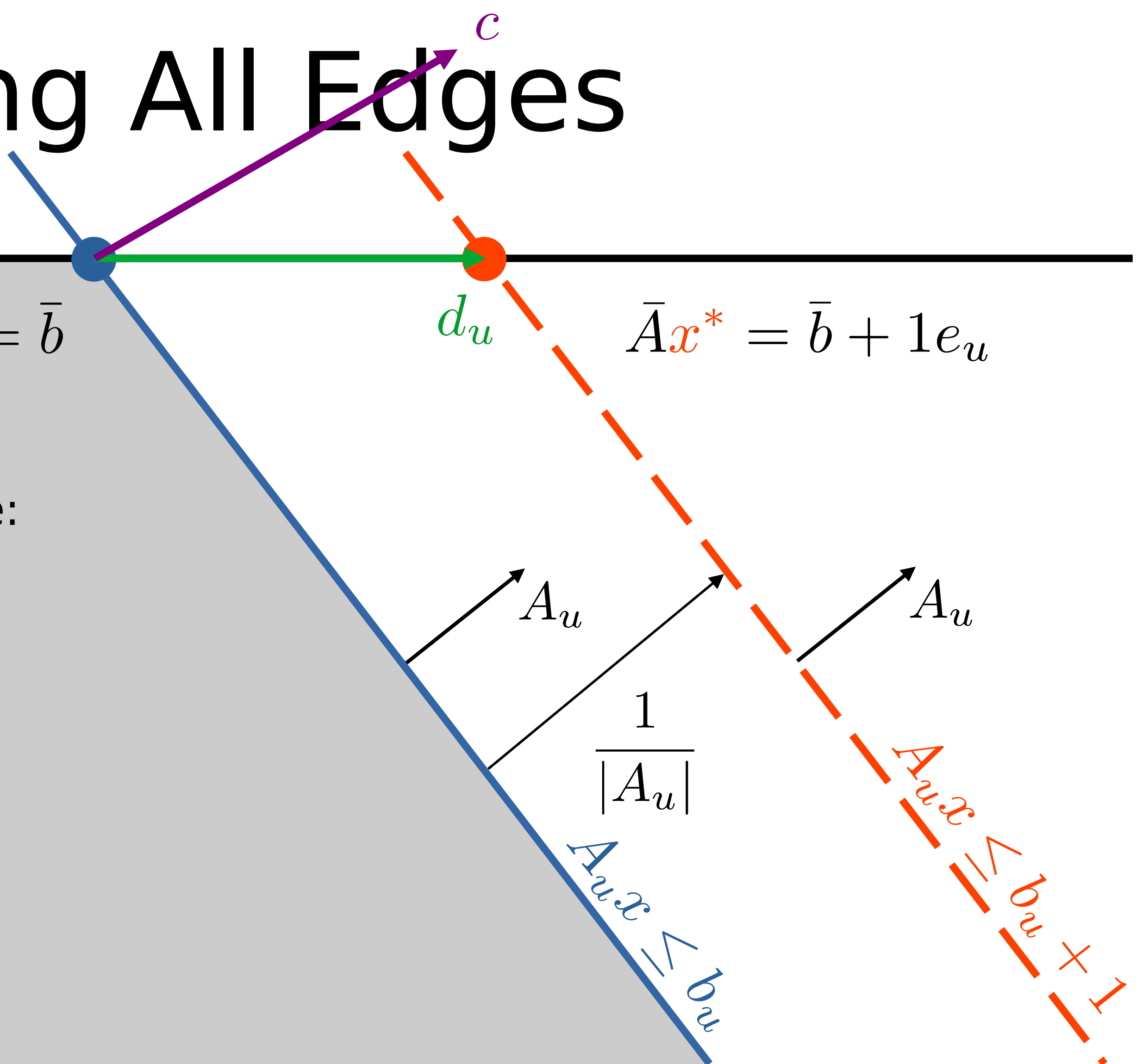
For all active constraints at once:

$$D \in \mathbb{R}^{n \times n} \quad \mu \in \mathbb{R}^n$$

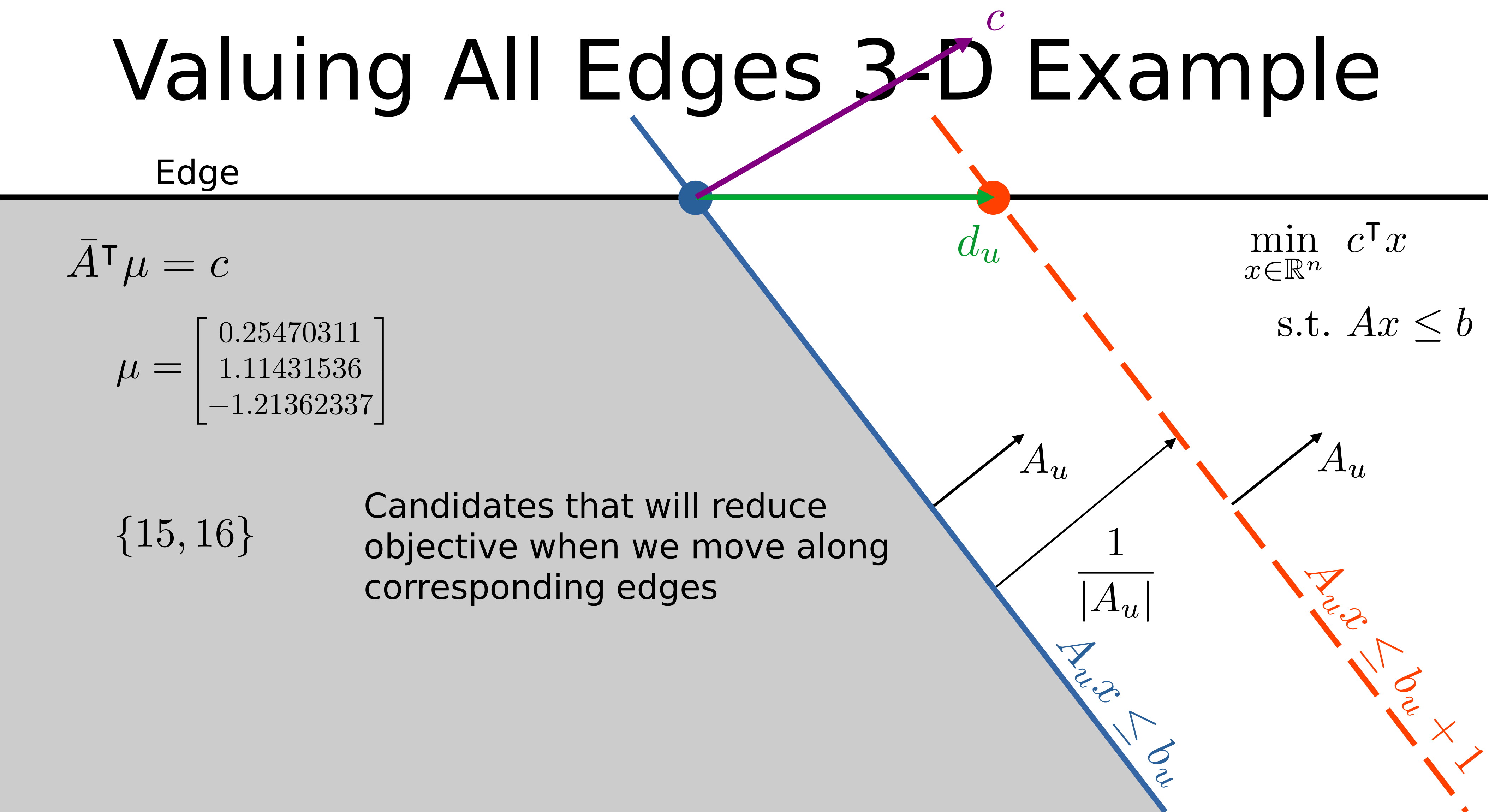
$$\bar{A} D = I \qquad \implies D = \bar{A}^{-1}$$

$$\mu = D^\intercal c = \bar{A}^{-\intercal} c$$

$$\implies \bar{A}^\intercal \mu = c$$



Valuing All Edges 3-D Example



$$\min_{x \in \mathbb{R}^n} \; c^\intercal x$$
$$\text{s.t. } Ax \le b$$

$$\bar{A}^\intercal \mu = c$$

$$\mu = \begin{bmatrix} 0.25470311 \\ 1.11431536 \\ -1.21362337 \end{bmatrix}$$

$$\{15, 16\}$$

Candidates that will reduce objective when we move along corresponding edges

Pivoting

The act of picking a **constraint to remove from the active set** (which edge to move along), and then picking a **constraint to replace it** .

$$\bar{A}^\intercal \mu = c$$

$$\min_{x \in \mathbb{R}^n} \; c^\intercal x$$

$$\text{s.t.} \; Ax \leq b$$

For a problem in form, any elements of μ that are positive represent constraints we could kick out of S_k.

What if max?

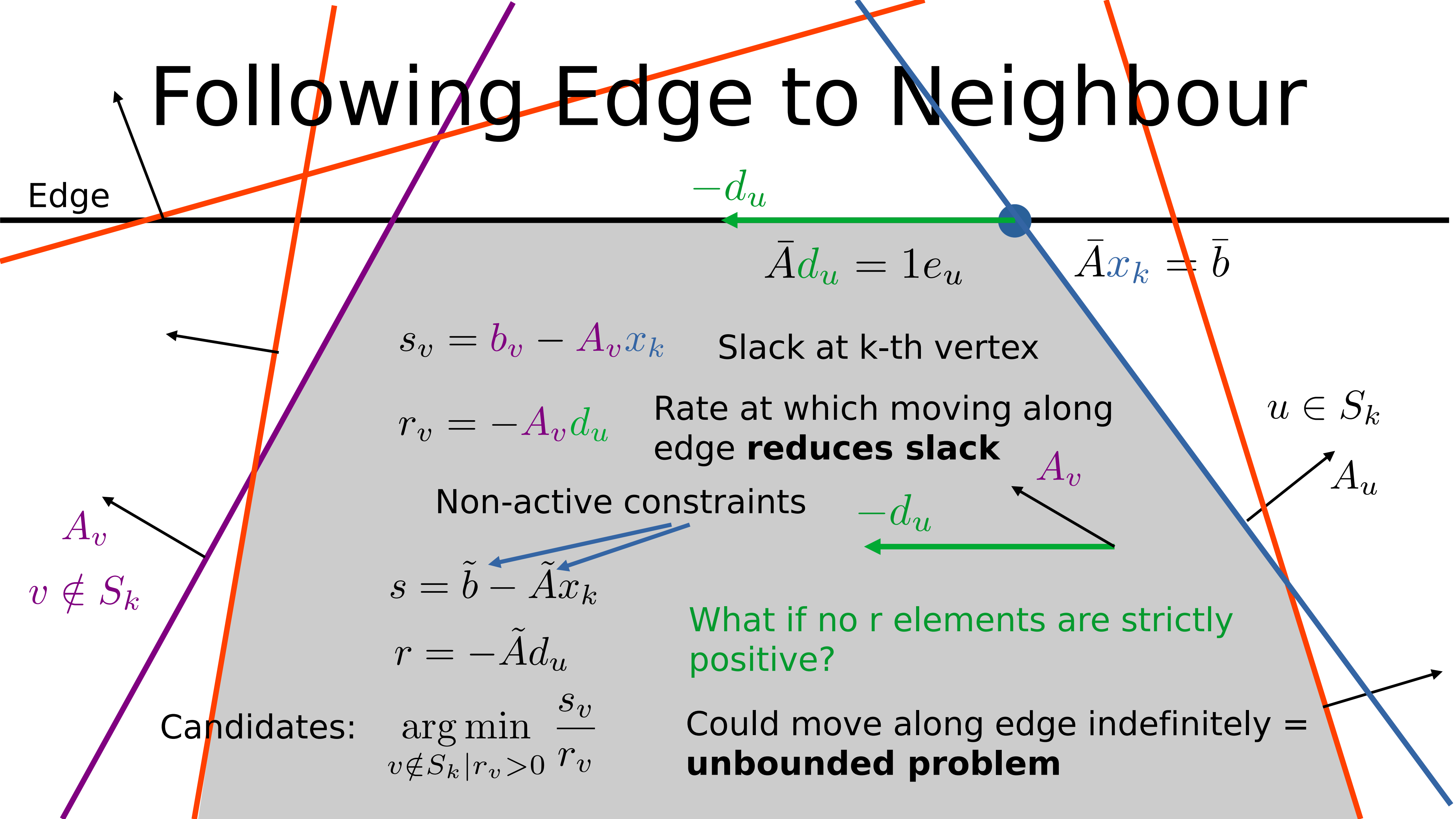
(conventions may vary)

What if none are positive?

- We have **the optimal value**, so can stop.

When there are multiple, then **pivot selection rules** are utilised, which are important for guaranteeing convergence when there is **degeneracy**.

Following Edge to Neighbour



$s_v = b_v - A_v x_k$ Slack at k-th vertex

$r_v = -A_v d_u$ Rate at which moving along edge **reduces slack**

Non-active constraints

$s = \tilde{b} - \tilde{A}x_k$

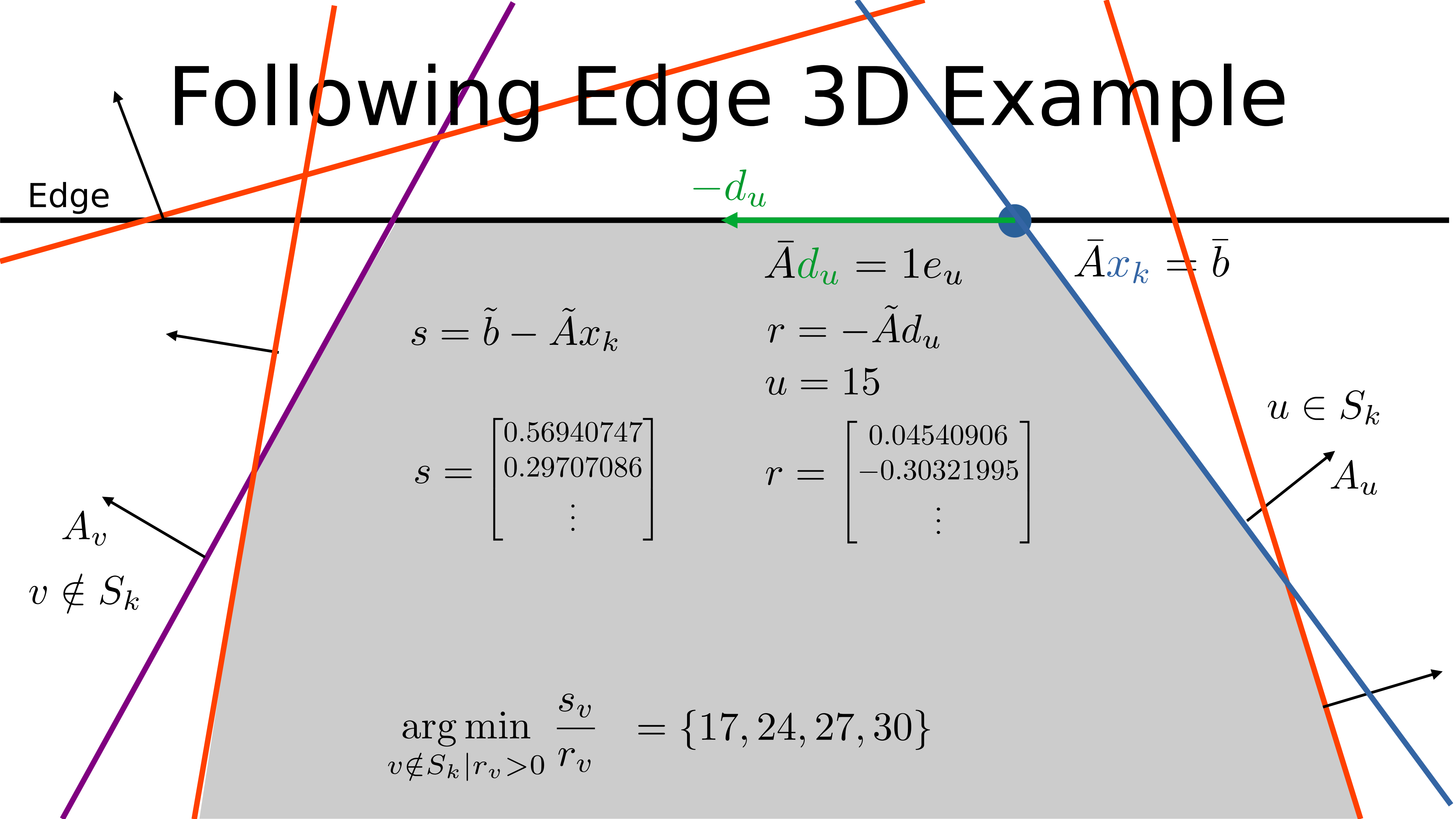
$r = -\tilde{A}d_u$

Candidates: $\underset{v \notin S_k | r_v > 0}{\arg\min} \dfrac{s_v}{r_v}$

What if no r elements are strictly positive?

Could move along edge indefinitely = **unbounded problem**

Following Edge 3D Example



Alternative Simplex Summary

Each $\boxed{}$ is a system of linear equations to be solved

Current active constraint set: S_k

Vertex position: $\boxed{\bar{A}x = \bar{b}}$

Pick leaving constraint: (otherwise optimal) $\boxed{\bar{A}^\intercal \mu = c} \Rightarrow u \in \{u | \mu_u > 0\}$

Pick joining constraint: (otherwise unbounded) $\boxed{\bar{A} d_u = 1 e_u} \Rightarrow \begin{array}{l} s = \tilde{b} - \tilde{A}x \\ r = -\tilde{A} d_u \end{array} \Rightarrow v \in \underset{v \notin S_k | r_v > 0}{\arg\min} \frac{s_v}{r_v}$

Next active constraint set: $S_{k+1} := (S_k \setminus \{u\}) \cup \{v\}$

Degenerate Vertices

Multiple **coincident vertices**. μ indicates it is worth pivoting, but we don't actually move anywhere because we immediately hit another constraint.

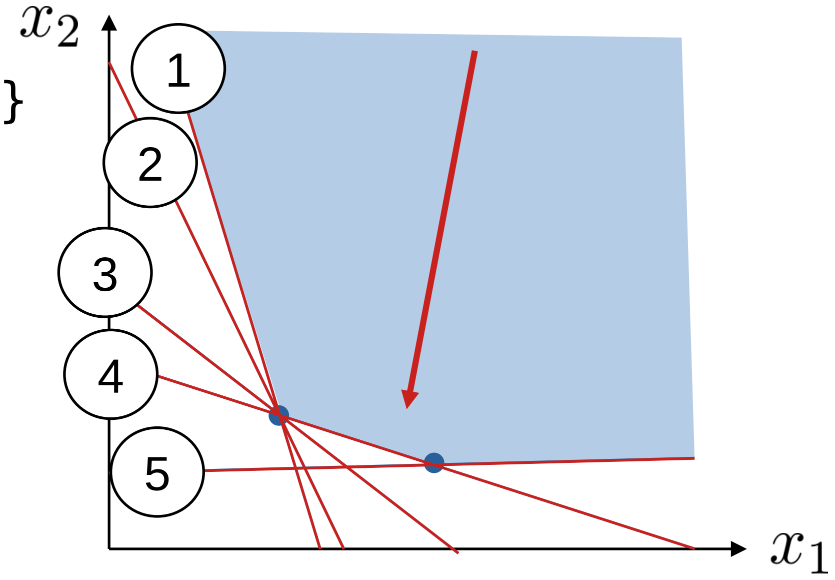
Why does it matter?

- It can slow down progress
- Certain pivot rules can enter into cycles and never converge

$S_1=\{1,2\} \rightarrow \{2,3\} \rightarrow \{1,3\} \rightarrow \{3,4\} \rightarrow \{4,5\}$

The **pivot rule is critical**, not just to avoid cycling but to achieve good performance, some options for selecting between candidates (where $\mu_u > 0$):

- The most positive μ_u
- Randomly (eventually avoids cycling)
- Try from smallest index (avoids cycling)



Finding a Starting Vertex

$$\min_{x \in \mathbb{R}^n} \quad c^\intercal x$$
$$\text{s.t.} \quad Ax \leq b$$

Solve a modified LP, with obvious starting vertex, whose optimal solution is a vertex of the original LP.

Call this new LP **the feasibility problem**.

Finding this first feasible point, potentially as difficult as then finding the optimal.

Relax constraints by introducing **slack variables**:

$$\min_{x \in \mathbb{R}^n, s \in \mathbb{R}^m} \quad 1^\intercal s$$
$$\text{s.t.} \quad Ax - s \leq b$$
$$s \geq 0$$

Optimal: x^*, s^*

If s* = 0, then $Ax^* \leq b$ i.e., vertex of the original!

Start the active constraint set for the original LP with a subset of n constraints where:

$$A_i x^* = b_i$$

What if s* ≠ 0? Original problem is infeasible!

Break the Chicken-Egg Cycle

We still need a starting vertex for the **feasibility problem...**

regarding A of the original problem

$$\begin{aligned} \min_{x \in \mathbb{R}^n, s \in \mathbb{R}^m} \quad & 1^\intercal s \\ \text{s.t.} \quad & Ax - s \leq b \\ & s \geq 0 \end{aligned}$$

Feasibility problem has n + m **variables**,
2m **constraints**.
(m > n most of the time)

Pick n linearly independent original constraints to be active.
Assume the first n:

$\forall i \in \{1, \ldots, n\}$

$A_i x = b_i \implies A_i x - s_i \leq b_i$

$s_i \geq 0$ **active**

2n active constraints,
need m - n more for a vertex

Solve: $A_{1:n} x^* = b_{1:n}$

$\forall i \in \{n+1, \ldots, m\}$

remaining constr of the original LP

	$s_i \geq 0$	$A_i x - s_i \leq b_i$
$A_i x^* > b_i$	inactive	active
$A_i x^* \leq b_i$	active	inactive

Complexity

Simplex Algorithm

No known pivot rule that can in general avoid visiting an exponential number of vertices.

As far as we know the **simplex algorithm** is worst-case exponential.

In practice for most problems it performs very well: often low-order polynomial number of pivots.

Ellipsoidal Algorithm

1979 Leonid Khachiyan proved that an ellipsoidal algorithm can solve any LP in polynomial time.

The problem of **solving a Linear Program is polynomial!**

In practice this algorithm performs poorly: higher degree polynomial for typical problems.

Interior-Point Algorithm

1947 John von Neumann (not polynomial time)

1984 Narendra Karmarkar developed an interior-point algorithm, Karmarkar's algorithm, for LPs that is polynomial time.

In practice this and more modern interior-point algorithms are competitive with simplex and don't have the worst-case exponential performance!

These two don't work with vertices, so avoid their combinatorial nature

Simplex Formulations

$$\min_{x \in \mathbb{R}^n} \quad c^\intercal x$$
$$\text{s.t.} \ Ax = b$$
$$x \geq 0$$

(m < n most of the time)

- **Standard** Simplex Algorithm: tableau manipulation
- **Revised** Simplex Algorithm: matrix manipulation

Work with **“basic” variables** in the “basis” set B_k , $|B_k| = m$

- B_k maps to a set of n active constraints
- All **m equalities always active**
  - **Need n – m inequalities to be active** (variables = 0)
- B_k represents the remaining set of n – (n – m) = m vars
  - only variables in B_k can be non-zero (inactive constr)

$$\min_{x \in \mathbb{R}^n} \quad c^\intercal x$$
$$\text{s.t.} \ Ax \leq b$$

✔ **“ Alternative”** Simplex Algorithm: matrix manipulation

Work with **active constraint** set S_k, $|S_k| = n$

Basic Feasible Solution (BFS)

Basic feasible solution (BFS): just another way of saying a vertex of the feasible polyhedron.

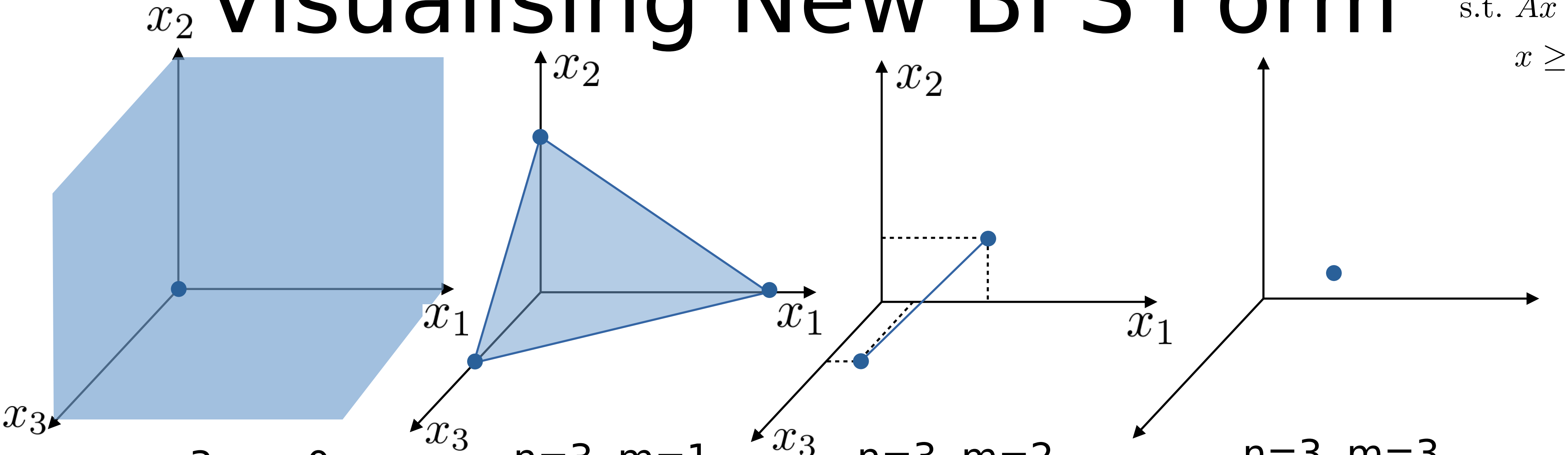
Basis: $B_k \subseteq \{1, \ldots, n\}, \quad |B_k| = m$

Basic variables: $x_j \geq 0 \quad \forall j \in B_k$

Non-basic variables: $x_j = 0 \quad \forall j \notin B_k$

Visualising New BFS Form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\intercal x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$



n=3, m=0

$B = \{\}$

n=3, m=1

$B = \{1\}$

$B = \{2\}$

$B = \{3\}$

n=3, m=2

$B = \{1, 2\}$

$B = \{1, 3\}$

n=3, m=3

$B = \{1, 2, 3\}$

Basis: $|B_k| = m$

Basic variables: $x_j \geq 0 \quad \forall j \in B_k$

Non-basic variables: $x_j = 0 \quad \forall j \notin B_k$

Revised Simplex Summary

Current BFS: B_k

Different decomposition over **vars not constr**: $\bar{A}\bar{x} + \tilde{A}\tilde{x} = b$

Basic

Non-basic

Vertex position: $\boxed{\bar{A}\bar{x} = b}$ $\tilde{x} = 0$

Pick entering variable: $\boxed{\bar{A}^{\mathsf{T}}\lambda = \bar{c}}$ $\Rightarrow$ $\mu = \tilde{c} - \tilde{A}^{\mathsf{T}}\lambda$ $\Rightarrow$ $u \in \{u | \mu_u < 0\}$

(otherwise optimal)

Pick leaving variable: $\boxed{\bar{A}d = A_{:,u}}$ $\Rightarrow$ $v \in \underset{v \in B_k | d_v > 0}{\arg\min} \frac{x_v}{d_v}$

(otherwise unbounded)

Next BFS: $B_{k+1} := (B_k \setminus \{v\}) \cup \{u\}$

Standard Simplex

Put coefficients of problem into a **"tableau"** (~a matrix), which is mutated by doing row operations to move from one BFS to the next.

Commonly taught, you should be aware it exists but we'll skip over it. It is OK for working small problems by hand, but implementation is less efficient than the matrix-based simplex formulations.

$$
\begin{aligned}
\min_{x,z} \quad & z \\
\text{s.t.} \quad & z - x_1 - 3x_2 = 0 \\
& x_1 + 2x_2 - 4x_3 + x_4 = 5 \\
& x_1 + 3x_3 - 2x_4 = 6 \\
& x \geq 0
\end{aligned}
\quad \longrightarrow \quad
\begin{array}{c}
\begin{array}{cccccc} z & x_1 & x_2 & x_3 & x_4 & \end{array} \\
\begin{bmatrix} 1 & -1 & -3 & 0 & 0 & 0 \\ 0 & 1 & 2 & -4 & 1 & 5 \\ 0 & 1 & 0 & 3 & -2 & 6 \end{bmatrix}
\end{array}
$$

Next

- LP Introduction
- Modelling and solving
- Feasible region and convexity
- Simplex algorithm
- **Relaxations and approximations**
  - Approximations
  - Convex relaxations
  - Battery scheduling example
  - Optimality gap
- **The dual of a linear program**