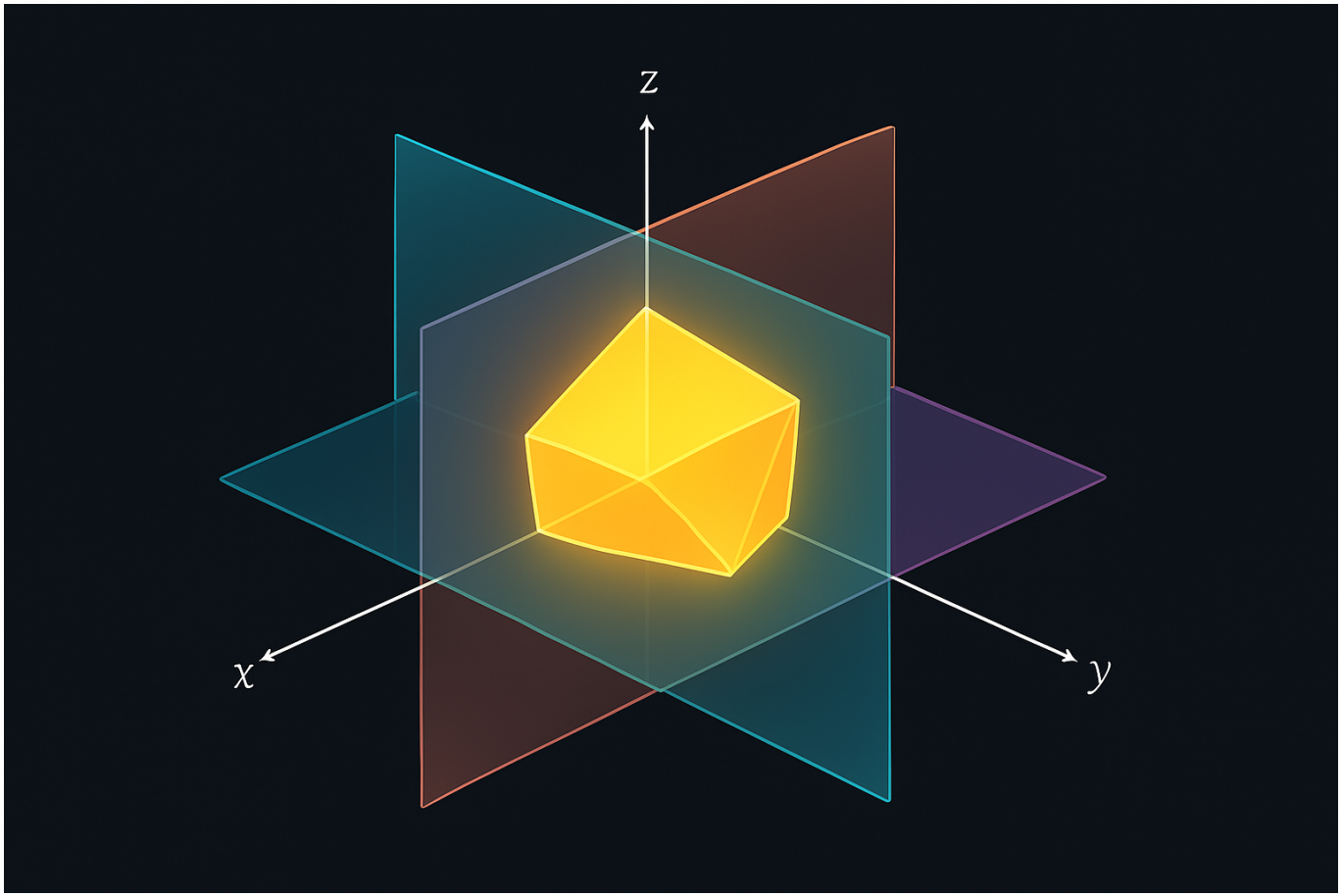
LP and Geometry



Deepak Ajwani

School of Computer Science

UCD DUBLIN

Overview

- *Linear Programming Basics*
- *Geometric View of LP*
- *Algorithms to Solve Linear Programmes (Coming Lectures)*
  - *Graphical method*
  - *Simplex Algorithm*
  - *Ellipsoid Algorithm*
  - *Karmarkar's Interior Point Method*

LP: reminder of the problem

- **WE ARE GIVEN:**
  - Set of *decision variables:* $x_1, x_2, \ldots, x_n$
  - *Objective function:* $c_1x_1 + c_2x_2 + \ldots + c_nx_n, \qquad c_i \in \mathbb{R}$
  - Set of *linear constraints:*

$$\begin{array}{rrrr} a_{11}x_1+ & a_{12}x_2+ & \ldots+ & a_{1n}x_n = b_1 \\ a_{21}x_1+ & a_{22}x_2+ & \ldots+ & a_{2n}x_n = b_2 \\ \vdots + & \vdots + & \ldots+ & \vdots = \vdots \\ a_{m1}x_1+ & a_{m2}x_2+ & \ldots+ & a_{mn}x_n = b_m \end{array}$$

$$x_j \geq 0 \ \textit{for all} \ j$$

LP: reminder of the problem

- **WE MUST FIND:** an assignment of values to $x_1, x_2, x_3, \ldots, x_n$ that *minimises* the objective function.

LP problems: Linear Algebra Notation

- In textbooks on linear programming, it is common to find a more compact notation, using only *matrices, matrix multiplication* and *vectors.*
- This notation comes from a branch of mathematics called *linear algebra.*
- During this module we'll stick to the longhand notation. The following slides are just to help you recognise the linear algebra notation if you ever come across it.

LP problems: Linear Algebra Notation

- We can write decision variables $x_1, x_2, \ldots, x_n$ as a single n-dimensional vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.
- Similarly we can write the cost factors $c_1, c_2, \ldots, c_n$ as a single *cost vector* $\mathbf{c} = (c_1, c_2, \ldots, c_n)$
- Then the objective function $c_1x_1 + c_2x_2 + c_3x_3 + \ldots + c_nx_n$ can be expressed mathematically as the *product* $\mathbf{c} \cdot \mathbf{x}$ of the 2 vectors

LP problems: Linear Algebra Notation

Notes:

- **c** · **x** is also known as the *dot product* or *inner product*

- Sometimes the "bar" $\bar{x}$ notation is used rather than boldface **x** to denote vectors

LP problems: Linear Algebra Notation

We can also reformulate the linear constraints into one single matrix equation.

$$
\begin{array}{rrcr}
a_{11}x_1+ & a_{12}x_2+ & \ldots+ & a_{1n}x_n = b_1 \\
a_{21}x_1+ & a_{22}x_2+ & \ldots+ & a_{2n}x_n = b_2 \\
\vdots\ + & \vdots\ + & \ldots+ & \vdots\ = \vdots \\
a_{m1}x_1+ & a_{m2}x_2+ & \ldots+ & a_{mn}x_n = b_m
\end{array}
$$

reformulates as

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$$

where **A** is the $m \times n$ matrix $[a_{ij}]$ and $\mathbf{b} = (b_1, b_2, \ldots, b_m)$.

LP problems: Linear Algebra Notation

Putting all this together, and not forgetting also the constraint that "$x_j \geq 0$ for all $j = 1, \ldots, n$", a linear program in linear algebra looks like this:

minimise $\mathbf{c} \cdot \mathbf{x}$
subject to the constraints:
$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$
$\mathbf{x} \geq 0$

Standard vs Canonical forms

- When LPs are given in this form, with constraints written as *equality* constraints

$$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n = b_i \qquad (i = 1,2,\ldots,m)$$

we say it is written in *standard form*.

- Sometimes the constraints are written as *inequality* constraints

$$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n \leq b_i \qquad (i = 1,2,\ldots,m)$$

in which case we say it is in *canonical form*.

Standard vs Canonical forms

Whether we adopt standard or canonical forms makes no difference in the following sense:

FACT For every LP written in standard form there exists an LP written in canonical form which has the same optimal solution, and vice versa.

From standard to canonical

Given an LP with equality constraints

$$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n = b_i \qquad (i = 1,2,\ldots,m)$$

QUESTION: How can we rewrite it in *canonical* form? (i.e., using only **in**equalities in terms of "≤"):

ANSWER: replace each constraint above by **2** inequality constraints

$$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n \leq b_i \qquad (i = 1,2,\ldots,m)$$

$$-a_{i1}x_1 - a_{i2}x_2 - \ldots a_{in}x_n \leq -b_i \qquad (i = 1,2,\ldots,m)$$

From canonical to standard

Given an LP with inequality constraints

$$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n \leq b_i \qquad (i = 1,2,\ldots,m)$$

QUESTION: How can we rewrite it in *standard* form? (i.e., using only equalities):

ANSWER: for each of the *m* constraints introduce a new decision variable s_i (called a *slack variable*), then replace the i^{th} constraint by the new constraint:

$$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n + s_i = b_i \qquad (i = 1,2,\ldots,m)$$

Canonical form

Can also replace “≤” by “≥” (and vice versa) if we wish:

Replace

$$a_{i1}x_1 + a_{i2}x_2 + \ldots a_{in}x_n \leq b_i$$

by

$$-a_{i1}x_1 - a_{i2}x_2 - \ldots a_{in}x_n \geq -b_i$$

Minimise or maximise?

- Our aim is often to *minimise* the objective function $c_1x_1 + c_2x_2 + c_3x_3 + \ldots c_nx_n$ (thinking in terms of *cost*)
- Sometimes we want to *maximise* it (thinking in terms of *utility*, or *profit*)
- Mathematically, it boils down to the same thing:

  the values of the decision variables that will *minimise* $c_1x_1 + c_2x_2 + c_3x_3 + \ldots c_nx_n$ are the same as those that *maximise* $-c_1x_1 - c_2x_2 - c_3x_3 - \ldots - c_nx_n$

The Geometry of Linear Programming

- Every linear program has a geometric interpretation
  - Translate the abstract language of algebra into the intuitive world of shapes and spaces.
- Why is the geometric interpretation important?
  - It gives intuition on how to go from symbols to a search
  - This connection is profound and all LP algorithms leverage it
  - It transforms the abstract task of solving a system of inequalities into the concrete, geometric task of:
    - Identifying the shape of the feasible region (the polytope)
    - Systematically searching through the polytope to find the optimal vertex

The Geometry of Linear Programming

- Variables → Dimensions
  - The number of variables in the LP (e.g., x_1, x_2) determines the number of dimensions of our geometric space (e.g., a 2D plane).
- A Linear Constraint → A Half-Space
  - Each linear inequality (e.g., $a_1x_1 + a_2x_2 \leq b$) defines a half-space.
  - This is the set of all points on one side of a line (in 2D) or a plane (in 3D).
- The Feasible Set → A Convex Polytope
  - The set of all points that satisfy *all* constraints is the feasible region.
  - Geometrically, this is the intersection of all the half-spaces. This intersection always forms a convex polygon (in 2D) or a convex polytope (in higher dimensions).
- The Optimal Solution → A Vertex
  - A fundamental theorem of linear programming states that if an optimal solution exists, it **must occur at a vertex** (a corner point) of the feasible polytope.

Desmos: a useful tool for doing the graphical method

- We can do the graphical method using *Desmos,* a web-based graphing calculator
  - https://www.desmos.com
- Also downloadable as an app for iOS and Android

In the tutorials, we will use Julia/JuMP for drawing the feasible region

Graphical method: An example

Consider the following LP

$$\textit{Maximise } 350x_1 + 300x_2$$

Subject to constraints:

$$x_1 + x_2 \leq 200$$

$$9x_1 + 6x_2 \leq 1566$$

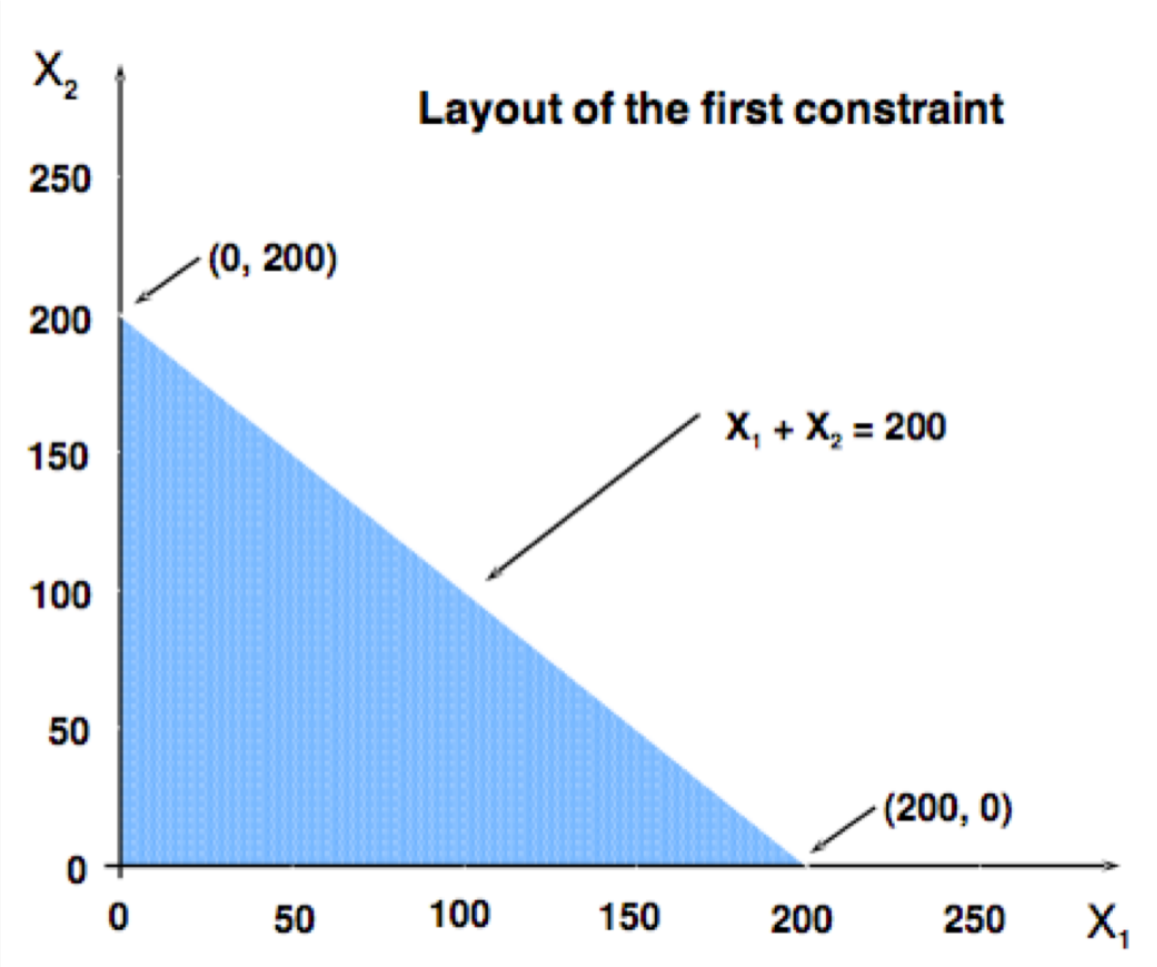
$$12x_1 + 16x_2 \leq 2880$$

$$x_1 \geq 0,\ x_2 \geq 0$$

What does the set of feasible solutions look like?

Let's add the constraints, one at a time.

The blue area is the set of possible points/solutions (x_1, x_2) that satisfy the first constraint $x_1 + x_2 \leq 200$ (together with $x_1 \geq 0,\ x_2 \geq 0$)

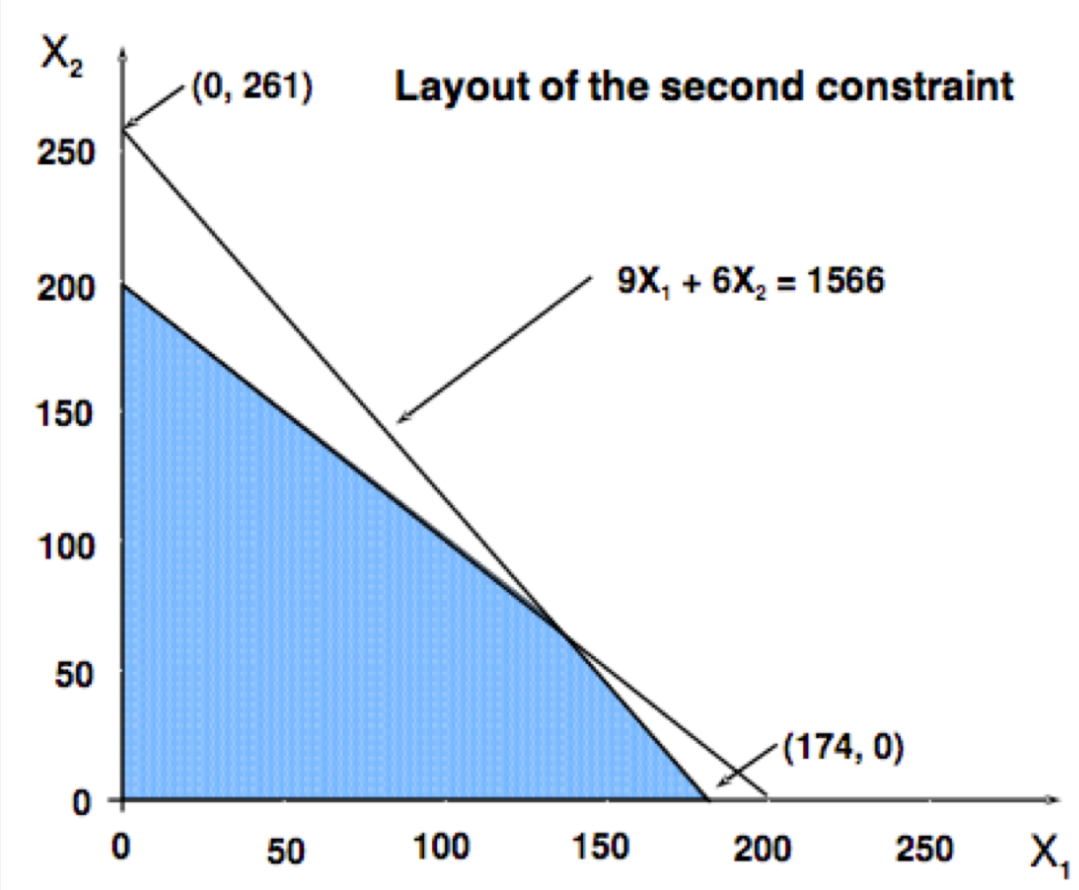


What does the set of feasible solutions look like?

Now add the second constraint

$$9x_1 + 6x_2 \leq 1566$$

The blue area is the set of possible points/solutions (x_1, x_2) that satisfy both the first 2 constraints (together with $x_1 \geq 0,\ x_2 \geq 0$)



What does the set of feasible solutions look like?

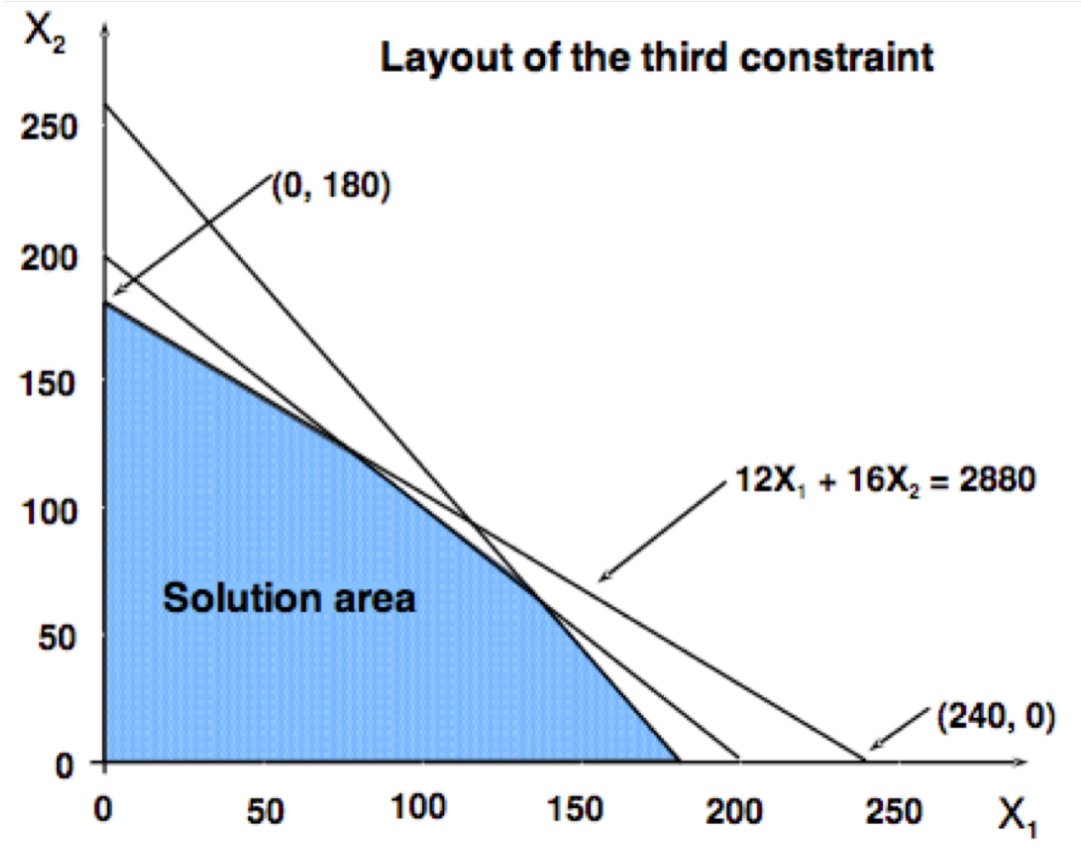
Finally add the third constraint

$$12x_1 + 16x_2 \leq 2880$$

The blue area is the set of possible points/solutions (x_1, x_2) for this LP

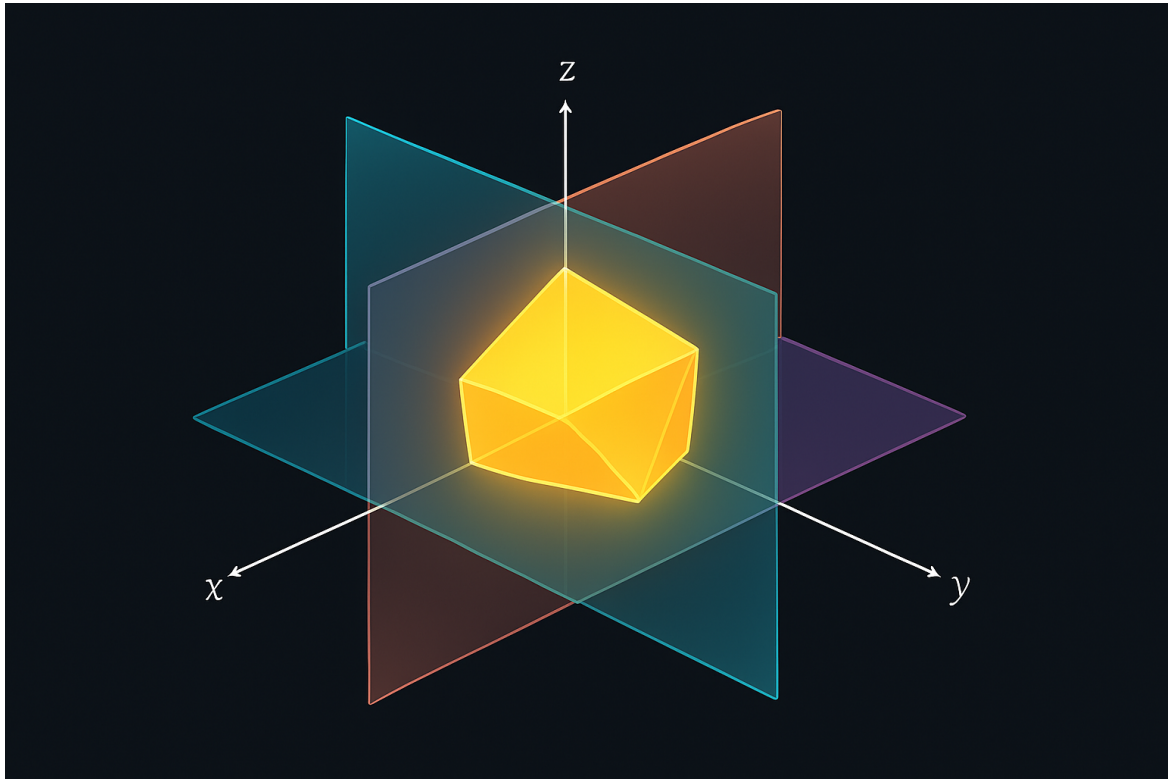
It forms a *polygon,* that is furthermore *convex*

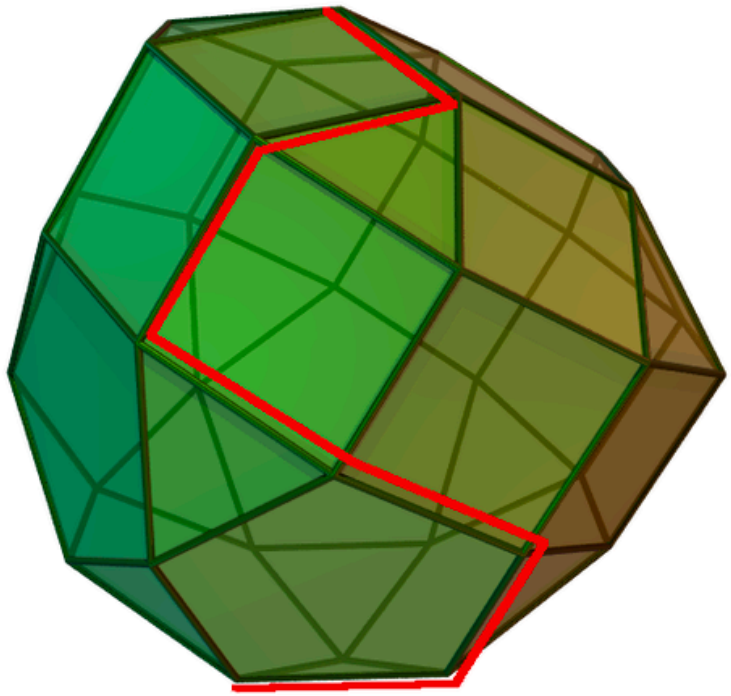
*Convex Polygon: T*ake any 2 points inside, then the straight line between them does not go outside the area



What about 3 decision variables?

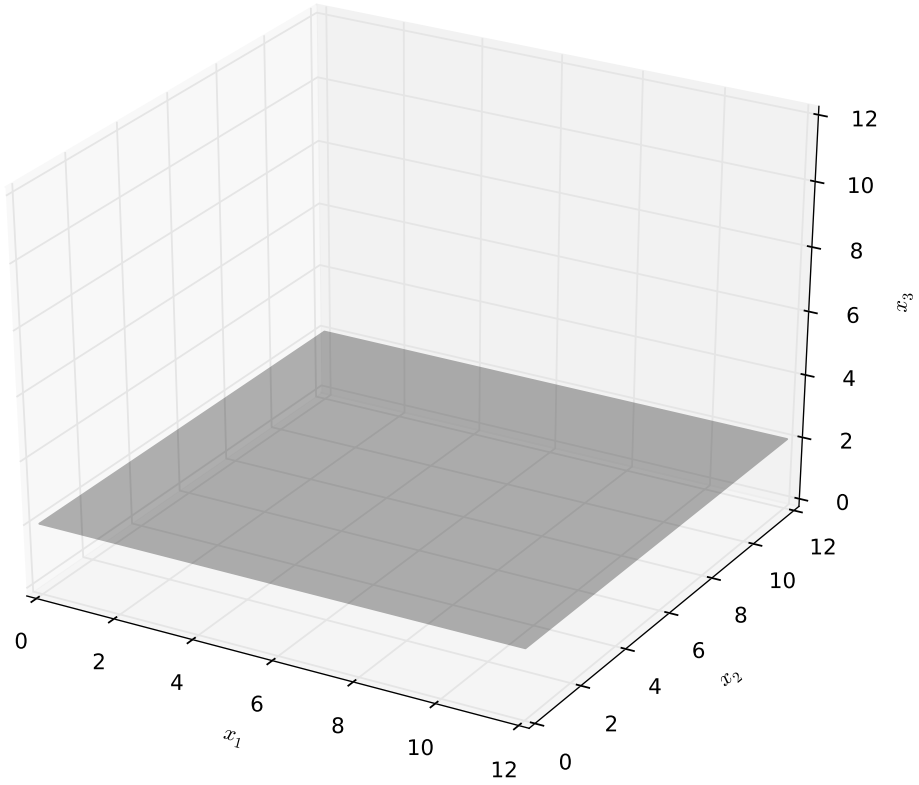
- Now each constraint corresponds to a *plane* rather than a line. And the feasible region is 3-dimensional.





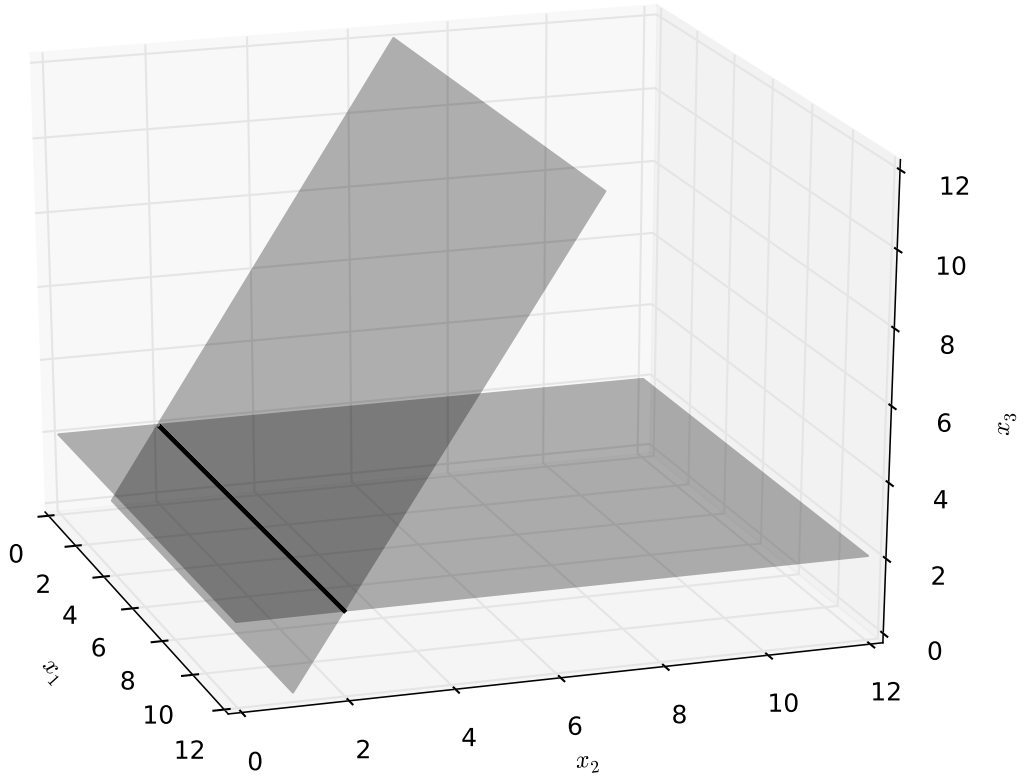
What about 3 decision variables?

- Set of basic feasible solutions given by set of vertices, as before.



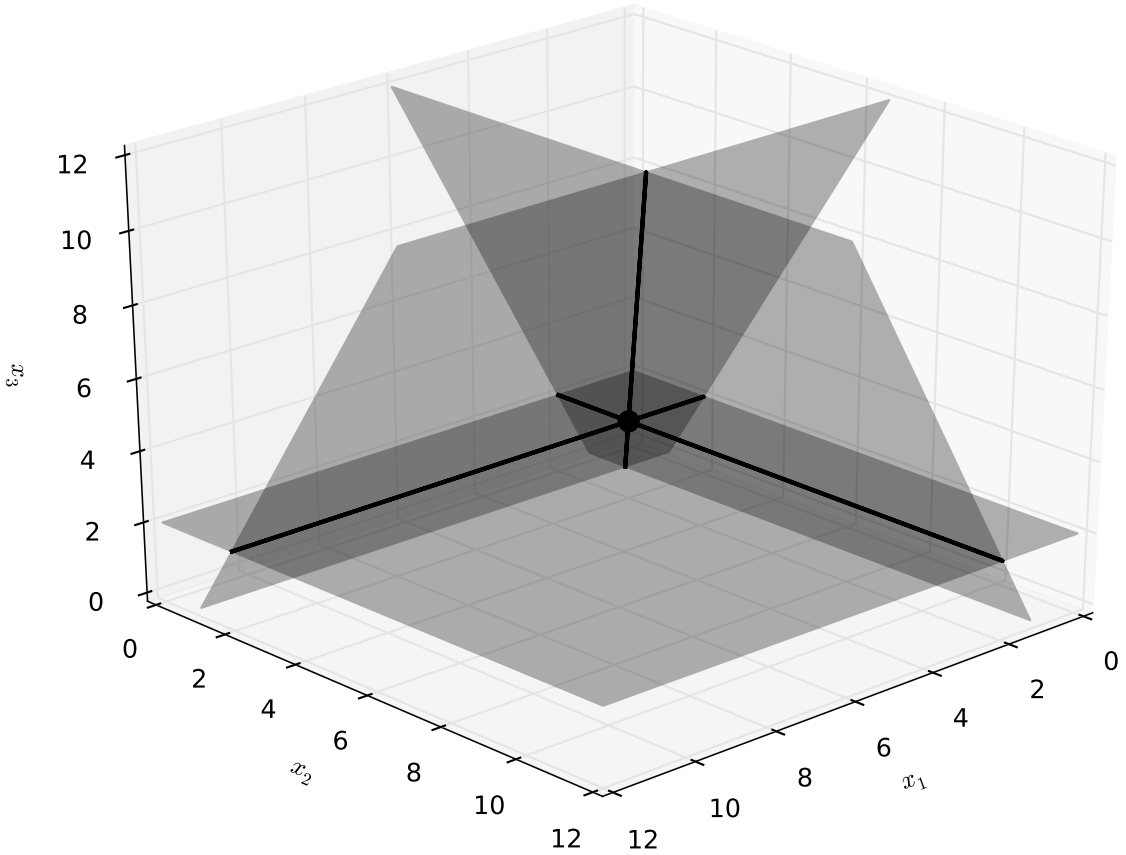
What about 3 decision variables?

- Set of basic feasible solutions given by set of vertices, as before.



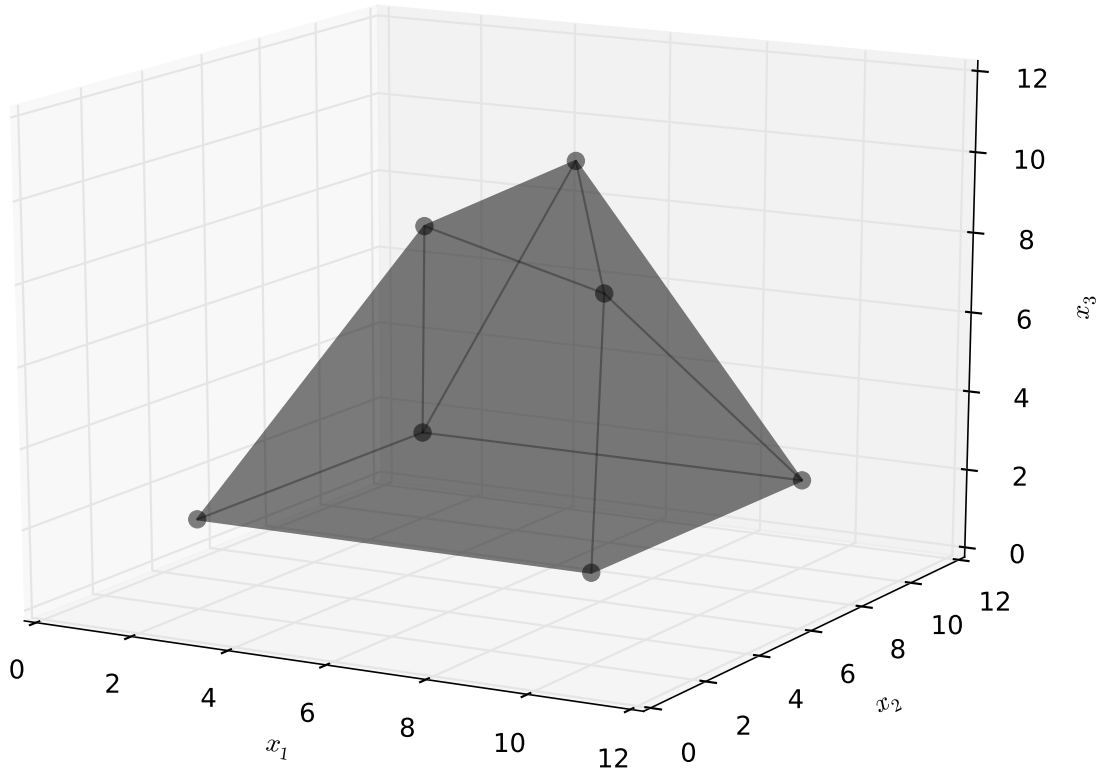
What about 3 decision variables?

- Set of basic feasible solutions given by set of vertices, as before.



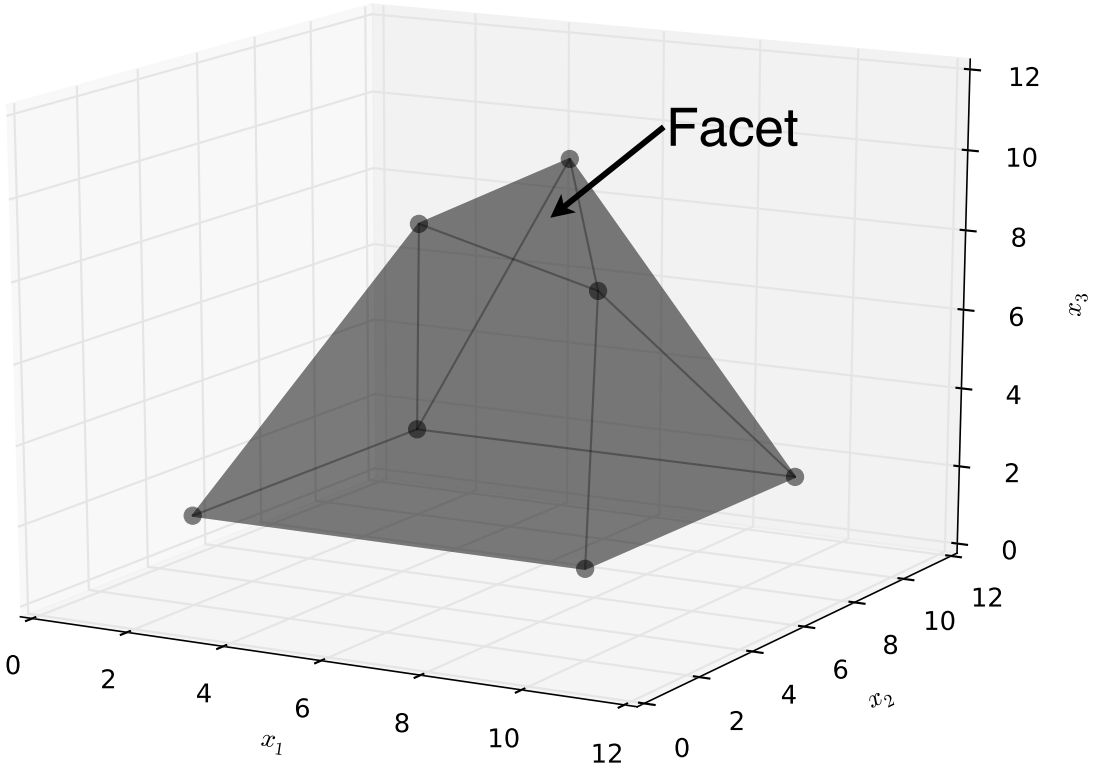
What about 3 decision variables?

- Set of basic feasible solutions given by set of vertices, as before.



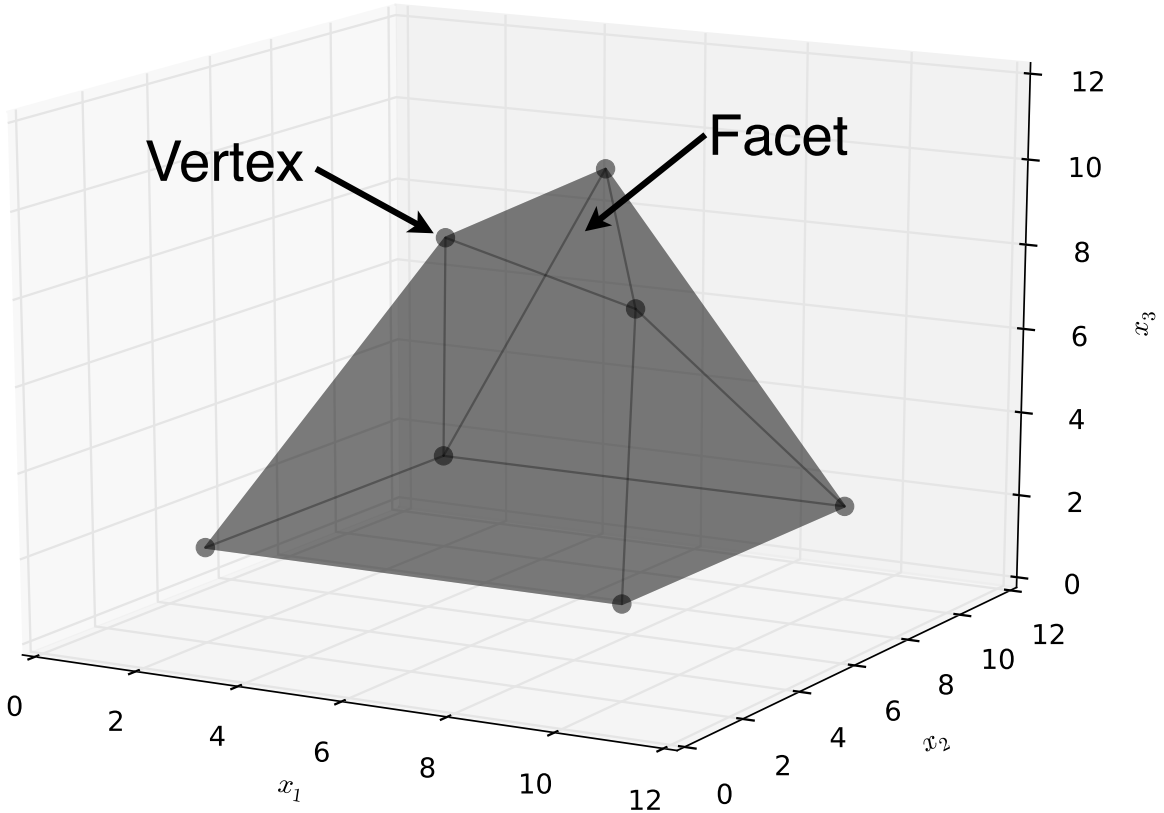
What about 3 decision variables?

- Set of basic feasible solutions given by set of vertices, as before.



What about 3 decision variables?

- Set of basic feasible solutions given by set of vertices, as before.



What about more decision variables?

- A bit harder to visualise!!
- But in principle the same interpretation applies. Instead of planes and polygons we speak of *hyperplanes* and *polytopes* (*n*-dimensional geometry).
- Feasible region is a higher dimensional polytope, but the optimal solution is still at a vertex
- We need to systematically traverse the feasible polytope to find the optimal vertex

Summary

- We looked at algebraic interpretation of linear programmes
- We considered a geometric interpretation of linear programmes
  - Set of feasible points usually (but not always) forms a *polygon* in 2-D space.
  - Vertices correspond to feasible solutions (basic feasible solutions)
  - An optimal solution (it it exists) can be found at one of these basic feasible solutions (or possibly at many of them)
- Optimisation algorithms will leverage both these interpretations to systematically explore the search space and find the optimal vertex