Lecture 4: Linear programming

- Brief recap: Linear algebra, Real analysis
- Linear programming, formulation, standard form
- KKT conditions for linear programming
- Dual problem, weak&strong duality

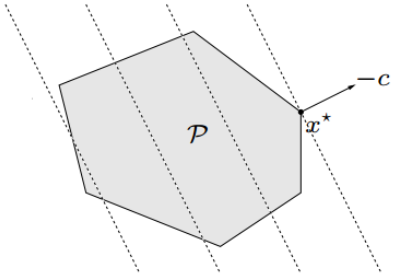
Reference: N&W Ch.13.1 (also: 12.9)

(Linear algebra: App A.1, Real analysis: App A.2)

Types of constrained optimization problems

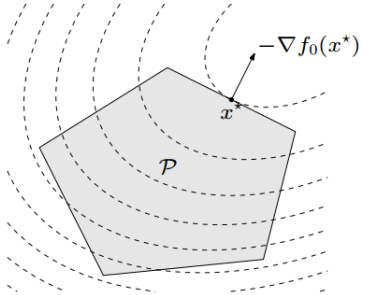
- Linear programming
  - Convex problem
  - Feasible set polyhedron

$$\begin{aligned} \text{minimize} \quad & c^\mathsf{T} x \\ \text{subject to} \quad & Ax \leq b \\ & Cx = d \end{aligned}$$



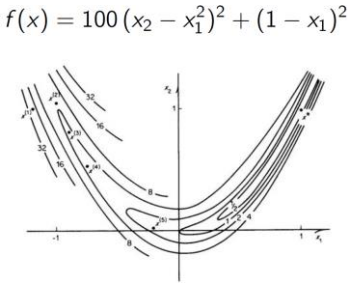
- Quadratic programming
  - Convex problem if $P \geq 0$
  - Feasible set polyhedron

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} x^\mathsf{T} P x + q^\mathsf{T} x \\ \text{subject to} \quad & Ax \leq b \\ & Cx = d \end{aligned}$$



- Nonlinear programming
  - In general non-convex!

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g(x) = 0 \\ & h(x) \geq 0 \end{aligned}$$



$$\min_{x \in \mathbb{R}^n} f(x) \qquad \text{subject to} \quad \begin{aligned} c_i(x) &= 0, \quad i \in \mathcal{E}, \\ c_i(x) &\geq 0, \quad i \in \mathcal{I}. \end{aligned}$$

from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

Defense Analyses put togeth-er a list they call the "Top Ten Algorithms of the Century."

"We tried to assemble the 10 al-gorithms with the greatest influence on the development and practice of science and engineering in the 20th century," Dongarra and Sullivan write. As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.

Without further ado, here's the CiSE top-10 list, in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer's reputation for deterministic calculation, it's fitting that one of its earliest applications was the generation of random numbers.

In terms of widespread use, George Dantzig's simplex method is among the most successful algorithms of all time.

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig's algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the "real" problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis at the National Bureau of Standards, initiate the development of **Krylov subspace iteration methods**.

These algorithms address the seemingly simple task of solving equations of the form $Ax = b$. The catch, of course, is that A is a huge $n \times n$ matrix, so that the algebraic answer $x = b/A$ is not so easy to compute.

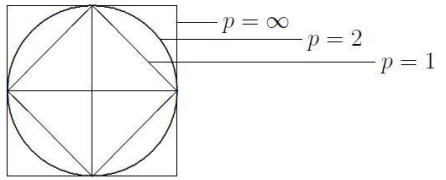
```
>> help norm
 norm   Matrix or vector norm.
     norm(X,2) returns the 2-norm of X.
     norm(X) is the same as norm(X,2).
     norm(X,1) returns the 1-norm of X.
     norm(X,Inf) returns the infinity norm of X.
     norm(X,'fro') returns the Frobenius norm of X.
```

Norms

- Vector norms:

  Vector norm: A mapping $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}^+$ that satisfies
  - $\|x\| = 0 \Rightarrow x = 0$
  - $\|x + z\| \leq \|x\| + \|z\|$, for all $x, z \in \mathbb{R}^n$
  - $\|\alpha x\| = |\alpha|\|x\|$, for all $\alpha \in \mathbb{R}$ and $x \in \mathbb{R}^n$

  Common norms (p-norms):
  - 1-norm: $\|x\|_1 = |x_1| + |x_2| + \ldots + |x_n|$ (sum norm)
  - 2-norm: $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$ (Euclidean norm
  - ∞-norm: $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$ (max norm)



$\|x\|_p = 1$

- Matrix norms (same definition as vector norm):

  Induced matrix norms, $A \in \mathbb{R}^{m \times n}$: $\|A\| := \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$
  - 1-norm: $\|A\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{m} |A_{ij}|$
  - 2-norm: $\|A\|_2 = \lambda_{\max}(\sqrt{A^\top A})$
  - ∞-norm: $\|A\|_\infty = \max_{i=1,\ldots,m} \sum_{j=1}^{n} |A_{ij}|$

  Other matrix norms, not induced:
  - Frobenius-norm $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2}$

  Useful property, induced matrix norms:
  - $\|Ax\| \leq \|A\|\|x\|$

$$\begin{pmatrix} 4 & -1 & 2 & 0 & 7 \\ 0 & 3 & 0 & 2 & 5 \\ -2 & 1 & 5 & 1 & 9 \\ 6 & 5 & 7 & 3 & \end{pmatrix} \quad \|A\|_\infty$$

$\|A\|_1$

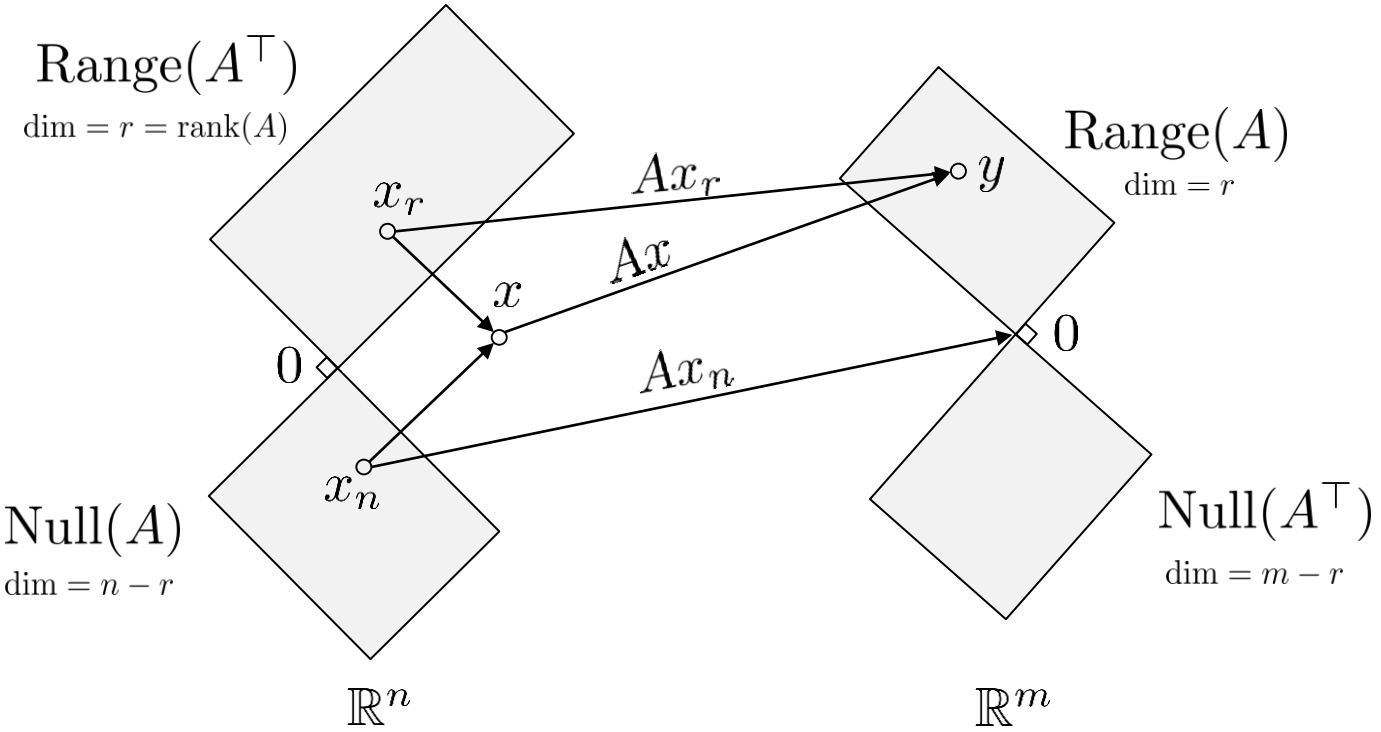
Fundamental theorem of linear algebra

A matrix $A \in \mathbb{R}^{m \times n}$ maps a vector $x \in \mathbb{R}^n$ into a vector $y \in \mathbb{R}^m$, $y = Ax$.

Nullspace of A: $\text{Null}(A) = \{w \mid Aw = 0\}$

Rangespace (columnspace) of A: $\text{Range}(A) = \{w \mid w = Av, \text{for some } v\}$

Fundamental theorem of linear algebra: $\text{Null}(A) \oplus \text{Range}(A^\top) = \mathbb{R}^n$



Condition number

- Well-conditioned: A small perturbation gives small changes

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3.00001 \\ 2 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.99999 \\ 1.00001 \end{pmatrix}$$

- Ill-conditioned: A small perturbation gives large changes

$$\begin{pmatrix} 1.00001 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2.00001 \\ 2 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1.00001 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \quad \Rightarrow \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$$

- Condition number:

$$\kappa(A) = \|A\|\|A^{-1}\|$$

  - A small condition number (say, 1-100) implies the matrix is well-conditioned, a large condition number (say, >10 000) implies the matrix is ill-conditioned. The condition number (2-norm) of the above matrices are 6.9 and 400 000, respectively.

```
>> help cond
 cond   Condition number with respect to inversion.
    cond(X) returns the 2-norm condition number (the
    ratio of the largest singular value of X to the smallest).
    Large condition numbers indicate a nearly singular
    matrix.

    cond(X,P) returns the condition number of X in P-norm:

       NORM(X,P) * NORM(INV(X),P).

    where P = 1, 2, inf, or 'fro'.
```

Matrix factorizations

Solve linear equation system:

$$Ax = b \quad \Rightarrow x = A^{-1}b$$

In practice, never use the inverse. It is inefficient and “unstable”.

Instead, use matrix factorizations:

- General matrix *A*: Use LU-decomposition (*Gaussian elimination*)

$$A = LU: \quad Ax = L\underbrace{Ux}_{y} = b \quad \Rightarrow \quad Ly = b \quad \Rightarrow \quad Ux = y$$

  - Due to triangular structure of *L* and *U*, we easily solve the two linear systems by substitution
- Symmetric pd matrix *A*: Use Cholesky decomposition

$$A = LL^\top$$

  - Half the cost of LU. Solve system as for LU.
  - For symmetric, indefinite matrices: Use LDL-factorization instead (book: $A = LBL^\top$)
- Generally, algorithms use permutations: $PA = LU, \quad PAP^\top = LL^\top$
- Other important factorizations
  - *A* = *QR*: Finds orthogonal basis for nullspace of *A*
  - Eigenvalue (spectral) decomposition, singular value decomposition

Some analysis (A.2)

Notation: $f(x) \in C^1$ means $f(x)$ once continuously differentiable, C^2 twice, etc.

- **Gradient and Hessian**: For a function $f : \mathbb{R}^n \to \mathbb{R}$, $f(x) \in C^2$, the gradient and Hessian are

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \quad \nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

- **Directional derivative**: The directional derivative of $f : \mathbb{R}^n \to \mathbb{R}$ is

$$D(f(x); p) := \lim_{\epsilon \to 0} \frac{f(x + \epsilon p) - f(x)}{\epsilon}$$

Also valid when $f(x)$ is not continuously differentiable. When $f(x) \in C^1$,

$$D(f(x); p) = \nabla f(x)^\top p$$

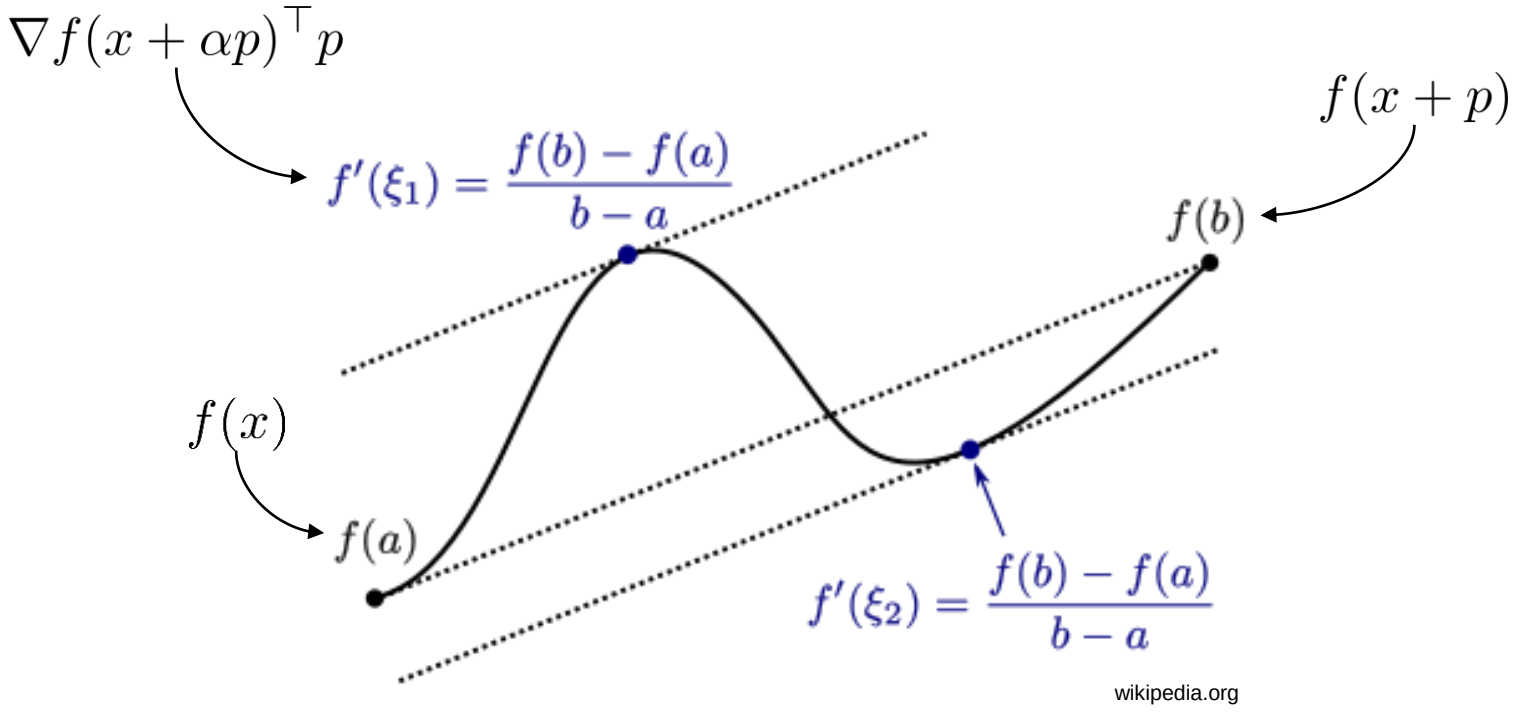
- **Lipschitz continuity**: A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is Lipschitz continuous in a neighborhood $\mathcal{N}$, if

$$\|f(x) - f(y)\| \leq L\|x - y\|, \quad \text{for all } x, y \in \mathcal{N}$$

Mean value theorem

- For $f : \mathbb{R}^n \to \mathbb{R}$, $f(x) \in C^1$, we have

$$f(x+p) = f(x) + \nabla f(x+\alpha p)^\top p \quad \text{for some } \alpha \in < 0, 1 >$$



wikipedia.org

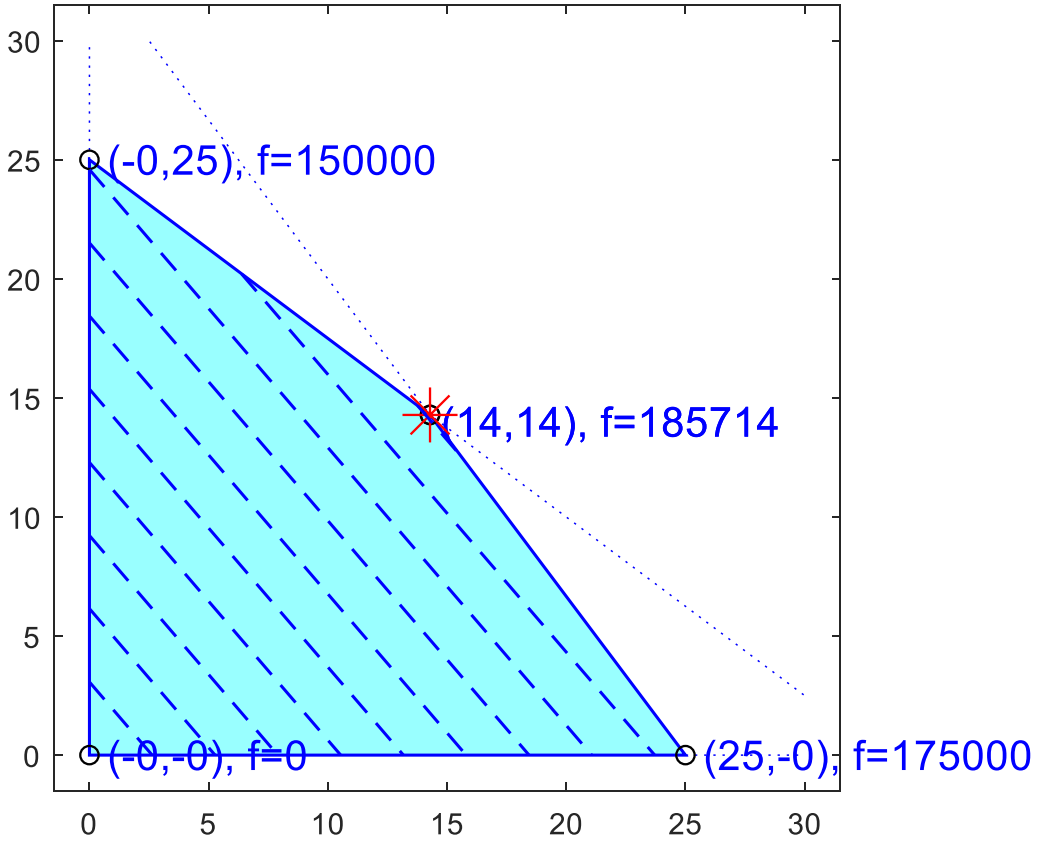
LP example: Farming

- A farmer wants to grow apples (A) and bananas (B)
- He has a field of size 100 000 m^2
- Growing 1 tonne of A requires an area of 4 000 m^2, growing 1 tonne of B requires an area of 3 000 m^2
- A requires 60 kg fertilizer per tonne grown, B requires 80 kg fertilizer per tonne grown
- The profit for A is 7000 per tonne (including fertilizer cost), the profit for B is 6000 per tonne (including fertilizer cost)
- The farmer can legally use up to 2000 kg of fertilizer

- He wants to maximize his profits

Farming example: Geometric interpretation and solution

$$\begin{aligned} \max_{x_1,\, x_2} \quad & 7000x_1 + 6000x_2 \\ \text{subject to:} \quad & 4000x_1 + 3000x_2 \leq 100000 \\ & 60x_1 + 80x_2 \leq 2000 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{aligned}$$



KKT conditions (Theorem 12.1)

Lagrangian: $$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

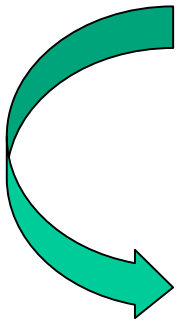
KKT-conditions (First-order necessary conditions): If x^* is a local solution and LICQ holds, then there exist λ^* such that

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \qquad \text{(stationarity)}$$

$$\left.\begin{aligned} c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \end{aligned}\right\} \qquad \text{(primal feasibility)}$$

$$\lambda_i^* \geq 0, \quad \forall i \in \mathcal{I}, \qquad \text{(dual feasibility)}$$

$$\lambda_i^* c_i(x^*) = 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \qquad \text{(complementarity condition/ complementary slackness)}$$



Either $\lambda_i^* = 0$ *or* $c_i(x^*) = 0$

Linear programming, standard form and KKT

LP, standard form: $\min_{x \in \mathbb{R}^n} c^T x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$

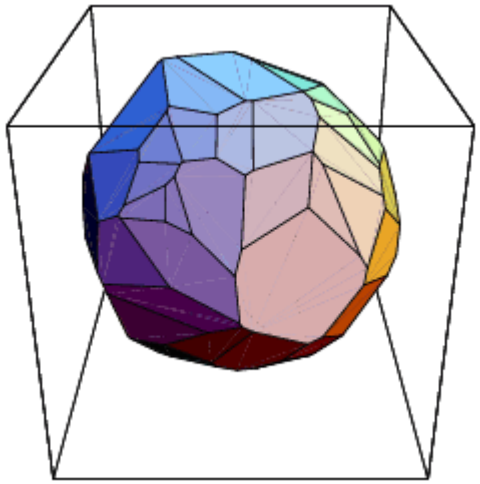
Lagrangian: $\mathcal{L}(x, \lambda, s) = c^T x - \lambda^T (Ax - b) - s^T x$

KKT-conditions (necessary *and* sufficient for LP):

$$\begin{aligned} A^T \lambda^* + s^* &= c, \\ Ax^* &= b, \\ x^* &\geq 0, \\ s^* &\geq 0, \\ x_i^* s_i^* &= 0, \quad i = 1, 2, \ldots, n \end{aligned}$$

Linear programming solutions

The feasible set is a polytope (a convex set with flat faces)



The objective function contours are planar

Three possible cases for solutions:

- No solutions: Feasible set is empty or problem is unbounded
- One solution: A vertex
- Infinite number of solutions: An “edge” is a solution