NTNU | Norwegian University of Science and Technology

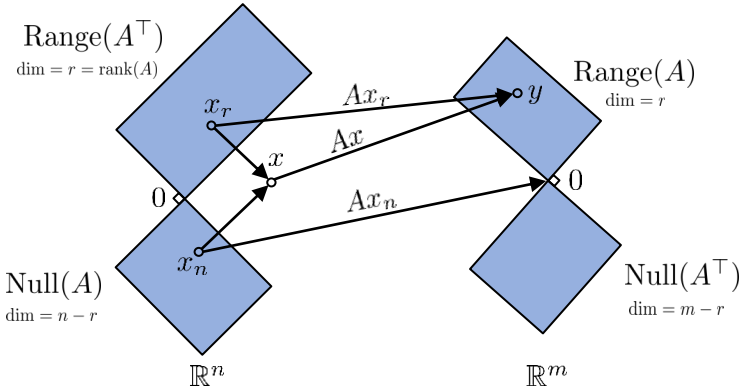
TTK4135 – Lecture 4
Linear programming

Lecturer: Lars Imsland

Purpose of Lecture

- Brief recap: Linear algebra
- Linear programming (LP): Formulation and standard form
- KKT conditions for LP
- The dual LP, weak & strong duality

Reference: Chapter 13.1 (12.9) in N&W (Linear algebra: A.1)



BRIEF RECAP: LINEAR ALGEBRA

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 $x \in \mathbb{R}^n$

Norms

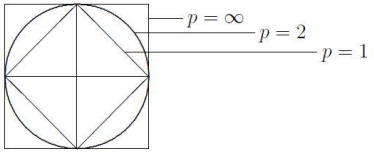
```
>> help norm
 norm   Matrix or vector norm.
     norm(X,2) returns the 2-norm of X.
     norm(X) is the same as norm(X,2).
     norm(X,1) returns the 1-norm of X.
     norm(X,Inf) returns the infinity norm of X.
     norm(X,'fro') returns the Frobenius norm of X.
```

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 vector 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)



$\|x\|_p = 1$

Norms

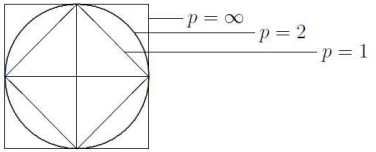
```
>> help norm
 norm   Matrix or vector norm.
     norm(X,2) returns the 2-norm of X.
     norm(X) is the same as norm(X,2).
     norm(X,1) returns the 1-norm of X.
     norm(X,Inf) returns the infinity norm of X.
     norm(X,'fro') returns the Frobenius norm of X.
```

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 vector 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)



$\|x\|_p = 1$

Induced matrix norms, $A \in \mathbb{R}^{m \times n}$: $\|A\|_p := \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$

- 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:

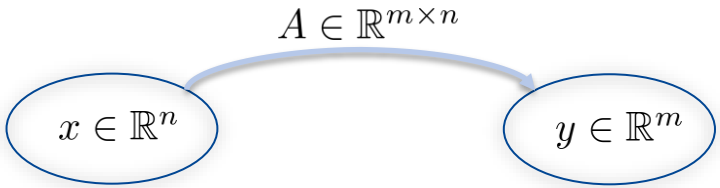
- $\|Ax\|_p \leq \|A\|_p \|x\|_p$

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

$\|A\|_1$

Fundamental Theorem of Linear Algebra

A matrix $A \in \mathbb{R}^{m \times n}$ is a mapping:



Nullspace of A: $\text{Null}(A) = \{v \in \mathbb{R}^n \mid Av = 0\}$

Rangespace (columnspace) of A: $\text{Range}(A) = \{w \in \mathbb{R}^m \mid w = Av, \text{ for some } v \in \mathbb{R}^n\}$

Fundamental Theorem of Linear Algebra

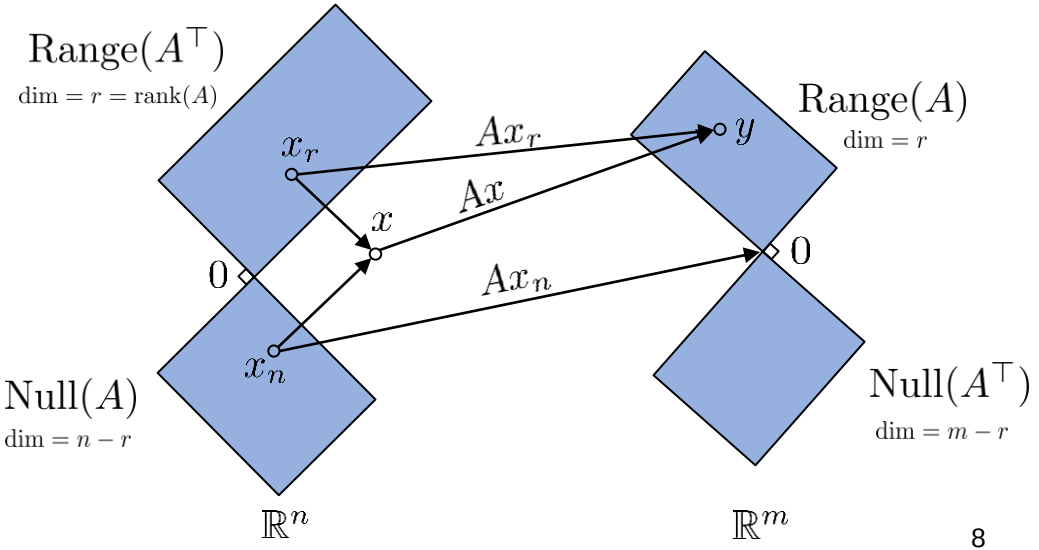
A matrix $A \in \mathbb{R}^{m \times n}$ is a mapping:

Nullspace of A: $\text{Null}(A) = \{x \in \mathbb{R}^n \mid Ax = 0\}$

Rangespace (columnspace) of A: $\text{Range}(A) = \{y \in \mathbb{R}^m \mid y = Ax, \text{ for some } x \in \mathbb{R}^n\}$

Fundamental theorem of linear algebra:

$$\text{Null}(A) \oplus \text{Range}(A^\top) = \mathbb{R}^n$$



Matrix Factorizations

```
>> help \
 \   Backslash or left matrix divide.
    A\B is the matrix division of A into B, which is roughly the
    same as INV(A)*B , except it is computed in a different way.
    If A is an N-by-N matrix and B is a column vector with N
    components, or a matrix with several such columns, then
    X = A\B is the solution to the equation A*X = B. A warning
    message is printed if A is badly scaled or nearly singular.
```

"All" algorithms in this course involve solving linear equation systems:

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

In practice, **never** use the matrix inverse. It is inefficient and numerically unstable.

Matrix Factorizations

```
>> help \
 \   Backslash or left matrix divide.
    A\B is the matrix division of A into B, which is roughly the
    same as INV(A)*B , except it is computed in a different way.
    If A is an N-by-N matrix and B is a column vector with N
    components, or a matrix with several such columns, then
    X = A\B is the solution to the equation A*X = B. A warning
    message is printed if A is badly scaled or nearly singular.
```

"All" algorithms in this course involve solving linear equation systems:

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

In practice, **never** use the matrix inverse. It is inefficient and numerically 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

Matrix Factorizations

```
>> help \
 \   Backslash or left matrix divide.
    A\B is the matrix division of A into B, which is roughly the
    same as INV(A)*B , except it is computed in a different way.
    If A is an N-by-N matrix and B is a column vector with N
    components, or a matrix with several such columns, then
    X = A\B is the solution to the equation A*X = B. A warning
    message is printed if A is badly scaled or nearly singular.
```

"All" algorithms in this course involve solving linear equation systems:

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

In practice, **never** use the matrix inverse. It is inefficient and numerically 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 positive definite 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: $A = LBL^\top$)

Matrix Factorizations

>> help \
\ Backslash or left matrix divide.
A\B is the matrix division of A into B, which is roughly the same as INV(A)*B , except it is computed in a different way. If A is an N-by-N matrix and B is a column vector with N components, or a matrix with several such columns, then X = A\B is the solution to the equation A*X = B. A warning message is printed if A is badly scaled or nearly singular.

"All" algorithms in this course involve solving linear equation systems:

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

In practice, **never** use the matrix inverse. It is inefficient and numerically 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 positive definite 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: $A = LBL^\top$)
- Other important factorizations
  - *A* = *QR*: Finds orthogonal (orthonormal) basis for rangespace of *A* and nullspace of A^T
  - Eigenvalue (spectral) decomposition, singular value decomposition

Note: Generally, algorithms use permutations:

$$PA = LU, \quad PAP^\top = LL^\top$$

Condition Number of a Matrix (when solving $Ax = b$)

- Well-conditioned matrix: Small perturbations give small changes in solution:

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

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

Condition Number of a Matrix (when solving $Ax = b$)

- Well-conditioned matrix: Small perturbations give small changes in solution:

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

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

- Ill-conditioned matrix: Small perturbations give large changes in solution:

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

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

Condition Number of a Matrix (when solving $Ax = b$)

- Well-conditioned matrix: Small perturbations give small changes in solution:

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

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

- Ill-conditioned matrix: Small perturbations give large changes in solution:

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

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

- 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.

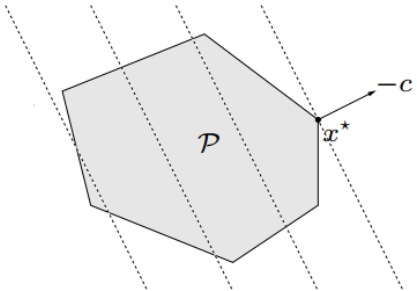
```
>> help cond
 cond   Condition number with respect to inversion.
    cond(X) returns the 2-norm condition number (the
    ratio of the largest singular value of X to the smallest).
    Large condition numbers indicate a nearly singular
    matrix.

    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'.
```

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} a_i^\top x - b_i = 0, & i \in \mathcal{E} \\ a_i^\top x - b_i \geq 0, & i \in \mathcal{I} \end{cases}$$



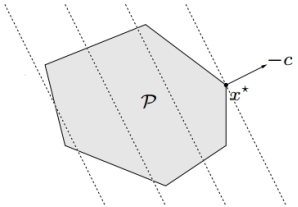
$$\begin{aligned} \min \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ & Cx = d \end{aligned}$$

LINEAR PROGRAMMING

Types of Constrained Optimization Problems

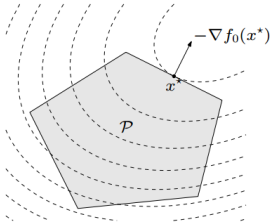
- Linear programming
  - Convex problem
  - Feasible set polyhedron

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



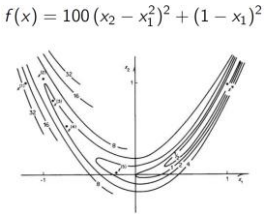
- Quadratic programming
  - Convex problem if $P \geq 0$
  - Feasible set polyhedron

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



- Nonlinear programming
  - In general non-convex!

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



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

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 wide-spread use, George Dantzig's simplex method is among the most successful al-gorithms 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**.

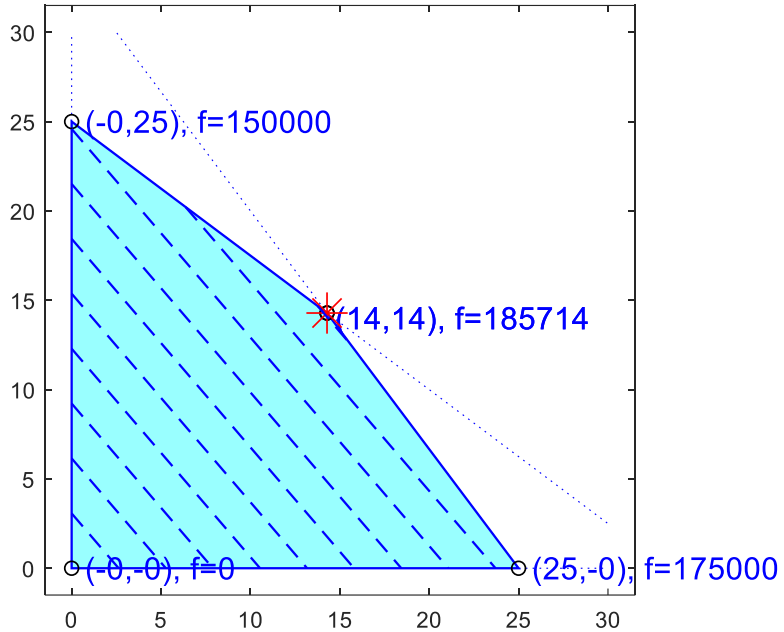
These algorithms address the seemingly simple task of solving equations of the form $Ax = b$. The catch, of course, is that A is a huge $n \times n$ matrix, so that the algebraic answer $x = b/A$ is not so easy to compute.

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

- The farmer wants to maximize his profits

Farming Example: Geometric Interpretation and Solution



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

Standard form LP

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Why standard form?

- More convenient to develop theory and algorithms for one form
- Standard form is particularly convenient for Simplex algorithm

- Good practice for students to think about transformations
  - Transformations are key techniques in modeling for optimization

How to transform into standard form:

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad Ax \leq b$$

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Number of variables and equations

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

We can assume without loss of generality that $A \in \mathbb{R}^{m \times n}$, $m < n$

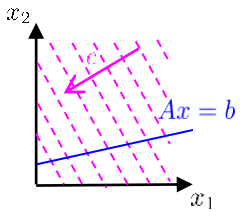
(If $m \geq n$, the problem is either infeasible, trivial, or can be transformed to an equivalent problem with $m < n$)

LPs does not always have a solution

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

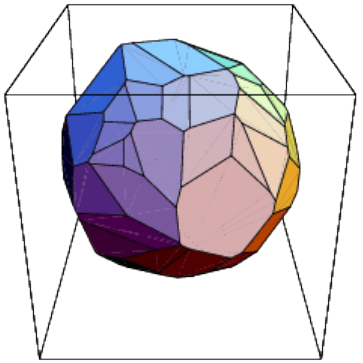
There are two sources for no solution:

1. Infeasibility: $Ax = b$ has no solution (the feasible set Ω is empty)
2. Unboundedness: There exists a sequence $x^k \in \Omega$ such that $c^\top x^k \to -\infty$



Linear Programming Solutions

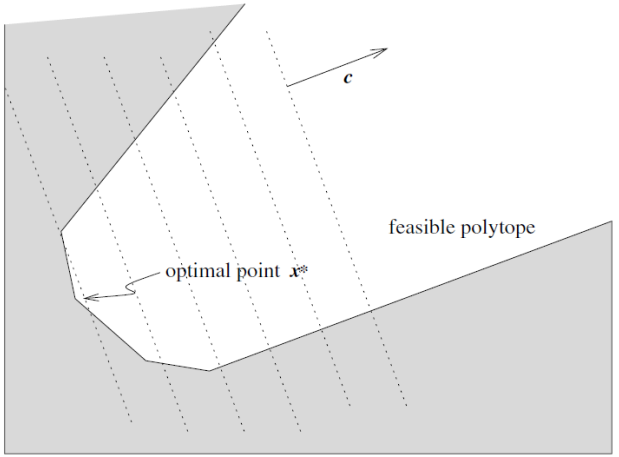
The feasible set is a polytope (a convex set with flat faces)



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



KKT Conditions for LPs

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Lagrangian:

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

KKT-conditions:

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\ c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\ c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{aligned}$$

KKT for LPs is necessary and sufficient

The dual LP problem

$$\max_{\lambda \in \mathbb{R}^m} b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \leq c$$

Lagrangian:

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

KKT-conditions:

$$\begin{aligned}
\nabla_x \mathcal{L}(x^*, \lambda^*) &= 0, \\
c_i(x^*) &= 0, \quad \forall i \in \mathcal{E}, \\
c_i(x^*) &\geq 0, \quad \forall i \in \mathcal{I}, \\
\lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I}, \\
\lambda_i^* c_i(x^*) &= 0, \quad \forall i \in \mathcal{E} \cup \mathcal{I}.
\end{aligned}$$

Weak duality: $c^\top \bar{x} \geq b^\top \bar{\lambda}$

Primal:

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Dual:

$$\max_{\lambda \in \mathbb{R}^m} b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \leq c$$

Strong duality

Theorem 13.1 Strong duality for LP

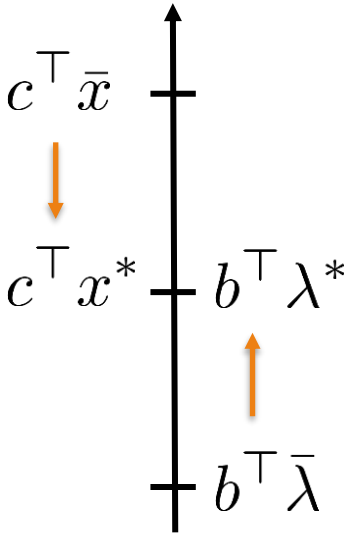
i) If primal or dual has a finite solution, so does the other, and $c^\top x^* = b^\top \lambda^*$

ii) If primal or dual is unbounded, the other is infeasible

Duality visualized

$$c^\top \bar{x} \geq c^\top x^* = b^\top \lambda^* \geq b^\top \bar{\lambda}$$

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$



$$\max_{\lambda \in \mathbb{R}^m} b^\top \lambda \quad \text{subject to} \quad A^\top \lambda \leq c$$

Sensitivity

- Given LP in standard form:

$$\min_{x \in \mathbb{R}^n} c^\top x \quad \text{subject to} \quad \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

- Assume optimal solution x^*, corresponding Lagrangian multiplier λ^*
- Do a small perturbation in b_i: $\tilde{b}_i = b_i + \epsilon$
- New solution fulfills

$$c^\top x^*_{\text{new}} = c^\top x^* \pm \epsilon \lambda^*_i$$