15-780: Linear Programming

J. Zico Kolter

February 1-3, 2016

Outline

Introduction

Some linear algebra review

Linear programming

Simplex algorithm

Duality and dual simplex

Outline

Decision-making with continuous variables

The problems we have focused on thus far in class have usually involved making *discrete* assignments to variables

An amazing property:

	Discrete search	**(Convex) optimization**
Variables	discrete	continuous real-valued
# possible solutions	finite	infinite
Complexity of solving	exponential	polynomial

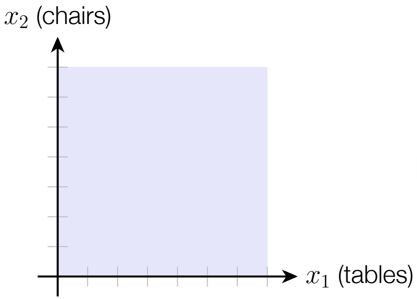
Techniques developed in Operations Research / Engineering communities, one of the biggest trends of the past ~15 years has been the integration into AI work

Example: manufacturing

A large factory makes tables and chairs. Each table returns a profit of \$200 and each chair a profit of \$100. Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood. The factory has 6K units of metal and 9K units of wood. How many tables and chairs should the factory make to maximize profit?

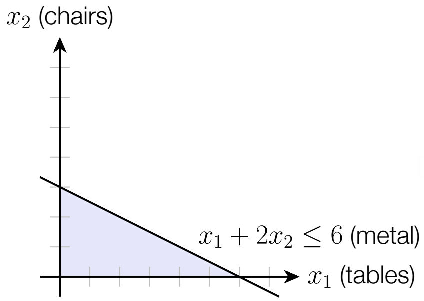
Example: manufacturing

A large factory makes tables and chairs. Each table returns a profit of \$200 and each chair a profit of \$100. Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood. The factory has 6K units of metal and 9K units of wood. How many tables and chairs should the factory make to maximize profit?



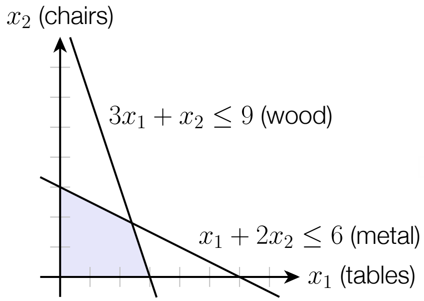
Example: manufacturing

A large factory makes tables and chairs. Each table returns a profit of \$200 and each chair a profit of \$100. Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood. The factory has 6K units of metal and 9K units of wood. How many tables and chairs should the factory make to maximize profit?



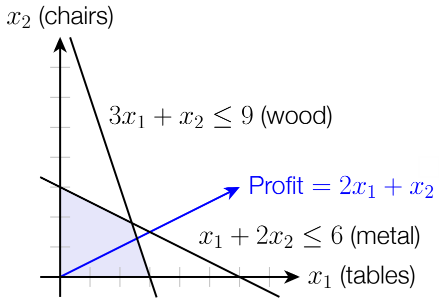
Example: manufacturing

A large factory makes tables and chairs. Each table returns a profit of $200 and each chair a profit of $100. Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood. The factory has 6K units of metal and 9K units of wood. How many tables and chairs should the factory make to maximize profit?



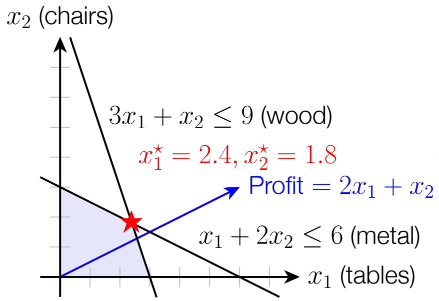
Example: manufacturing

A large factory makes tables and chairs. Each table returns a profit of $200 and each chair a profit of $100. Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood. The factory has 6K units of metal and 9K units of wood. How many tables and chairs should the factory make to maximize profit?



Example: manufacturing

A large factory makes tables and chairs. Each table returns a profit of \$200 and each chair a profit of \$100. Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood. The factory has 6K units of metal and 9K units of wood. How many tables and chairs should the factory make to maximize profit?



Example: manufacturing

A large factory makes tables and chairs. Each table returns a profit of \$200 and each chair a profit of \$100. Each table takes 1 unit of metal and 3 units of wood and each chair takes 2 units of metal and 1 unit of wood. The factory has 6K units of metal and 9K units of wood. How many tables and chairs should the factory make to maximize profit?

$$\begin{aligned} \underset{x_1, x_2}{\text{maximize}} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 6 \\ & 3x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Other examples

1. Solve tree-structured CSPs
2. Finding optimal strategies for two player, zero sum games
3. Finding most probable assignment in a graphical model
4. Finding (or approximating) solution of a Markov decision problem
5. Min-cut / max-flow network problems
6. Applications: economic portfolio optimization, robotic control, scheduling generation in smart grids, many many others

Comments

Not covered at all in textbook, lecture notes + video + supplementary material on class page is enough to complete all assignments

But, if you want a more thorough reference, you can look at: Bertsimas and Tsitsiklis, “Introduction to Linear Optimization”, Chapters 1-5.

Part of a much broader class of “tractable” optimization problems: *convex* optimization problems (we’ll return to more general problems and other optimization techniques later in the course)

Outline

Linear algebra

Throughout this lecture we're going to use matrix and vector notation to describe linear programming problems

This is very basic linear algebra (nothing more complex than addition, multiplication, and inverses)

But, if you're not familiar with linear algebra notation, it will be confusing, you can take a look at these lectures for some review:

`http://www.cs.cmu.edu/~zkolter/course/linalg`

Vectors

We use the notation $x \in \mathbb{R}^n$ to denote a vector with n entries

x_i denotes the ith element of x

By convention, $x \in \mathbb{R}^n$ represents a *column* vector (a matrix with one column and n rows); to indicate a *row* vector, we use x^T (x transpose)

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad x^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$$

Matrices

We use the notation $A \in \mathbb{R}^{m \times n}$ to denote a matrix with m rows and n columns

We use a_{ij} (or sometimes A_{ij}, $a_{i,j}$, etc) to denote the entry in the ith row and jth column; we use a_i to refer to the ith *column* of A, and a_i^T to refer to the ith *row* of A

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} | & & | \\ a_1 & \cdots & a_n \\ | & & | \end{bmatrix} = \begin{bmatrix} - & a_1^T & - \\ & \vdots & \\ - & a_m^T & - \end{bmatrix}$$

Addition/subtraction and transposes

Addition and subtraction of matrices and vectors is defined just as addition or subtraction of the elements, for $A, B \in \mathbb{R}^{m \times n}$,

$$C \in \mathbb{R}^{m \times n} = A + B \iff c_{ij} = a_{ij} + b_{ij}$$

Transpose operation switches rows and columns

$$D \in \mathbb{R}^{n \times m} = A^T \iff d_{ij} = a_{ji}$$

Matrix multiplication

For two matrices $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{m \times p}$,

$$C \in \mathbb{R}^{n \times p} = AB \iff c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Special case: *inner product* of two vectors, for $x, y \in \mathbb{R}^n$

$$x^T y \in \mathbb{R} = \sum_{i=1}^{n} x_i y_i$$

Special case: product of matrix and vector, for $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$

$$Ax \in \mathbb{R}^m = \begin{bmatrix} a_1^T x \\ \vdots \\ a_m^T x \end{bmatrix} = \sum_{i=1}^{n} a_i x_i$$

Matrix multiplication properties

Associative: $A(BC) = (AB)C$

Distributive: $A(B + C) = AB + AC$

Not commutative: $AB \neq BA$

Transpose of product: $(AB)^T = B^T A^T$

Matrix inverse

Identity matrix $I \in \mathbb{R}^{n \times n}$ is a square matrix with ones on diagonal and zeros elsewhere; has property that for any $A \in \mathbb{R}^{m \times n}$

$$IA = A, \;\; AI = A, \;\; \text{(for different sized } I \text{ matrices)}$$

For a *square* matrix $A \in \mathbb{R}^{n \times n}$, *inverse* A^{-1} is th matrix such that

$$A^{-1}A = I = AA^{-1}$$

Properties:

- May not exists even for square matrices (linear independence and rank conditions)

- $(AB)^{-1} = B^{-1}A^{-1}$ *if* A *and* B *are square and invertible*

- $(A^{-1})^T = (A^T)^{-1} \equiv A^{-T}$

Vector norms

For $x \in \mathbb{R}^n$, we'll use $\|x\|_2$ to denote the *Euclidean norm* of x

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x^T x}$$

(and also) $\|x\|_2^2 = \sum_{i=1}^{n} x_i^2 = x^T x$

Thus, for two vectors $x, y \in \mathbb{R}^n$, $\|x - y\|_2$ denotes the Euclidean distance between x and y

$$\|x - y\|_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} = \sqrt{x^T x + y^T y - 2x^T y}$$

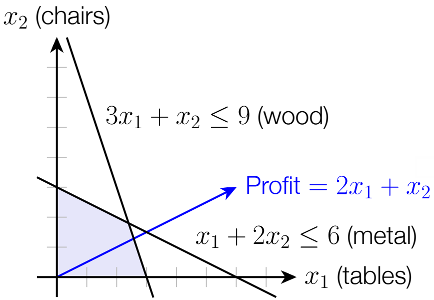
The 2 subscript denotes the fact that this is called the 2-norm of a vector, we'll see other norms like the 1-norm and ∞-norm

Outline

Example: manufacturing

Let's return to the example we started with:

$$\begin{aligned} \underset{x_1, x_2}{\text{maximize}} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \le 6 \\ & 3x_1 + x_2 \le 9 \\ & x_1, x_2 \ge 0 \end{aligned}$$



Inequality form linear programs

Using linear algebra notation, we can write linear program more compactly as

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & c^T x \\ \text{subject to} \quad & Gx \leq h \end{aligned}$$

with optimization variable $x \in \mathbb{R}^n$ and problem data $c \in \mathbb{R}^n$, $G \in \mathbb{R}^{m \times n}$, and $h \in \mathbb{R}^m$ and where the $\leq$ denotes elementwise inequality (equality constraints also possible: $g_i^T x = h_i \equiv g_i^T x \leq h_i, -g_i^T x \leq h_i$)

Example in inequality form:

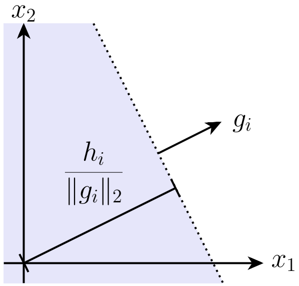
$$\begin{aligned} \underset{x_1, x_2}{\text{maximize}} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 6 \\ & 3x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0 \end{aligned} \qquad c = \begin{bmatrix} 2 \\ 1 \end{bmatrix} G = \begin{bmatrix} 1 & 2 \\ 3 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} h = \begin{bmatrix} 6 \\ 9 \\ 0 \\ 0 \end{bmatrix}$$

Geometry of linear programs

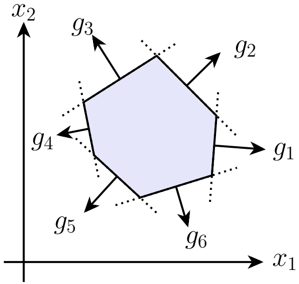
Consider the inequality constraints of the linear program written out explicitly

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & c^T x \\ \text{subject to} \quad & g_i^T x \leq h_i, \quad i = 1, \ldots, m \end{aligned}$$

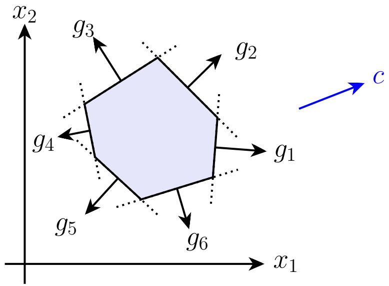
Each constraint $g_i^T x \leq h_i$ represents a *halfspace* constraint



Multiple halfspace constraints, $g_i^T x \leq h_i$, $i = 1, \ldots, m$ (or equivalently $Gx \leq h$), define what is called a *polytope*



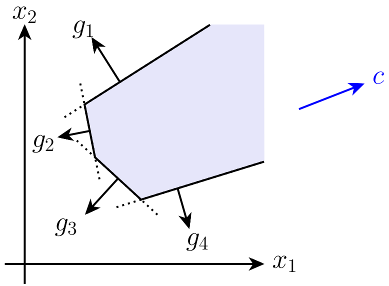
So linear programming is equivalent to maximizing some direction ($c^T x$) over a polytope



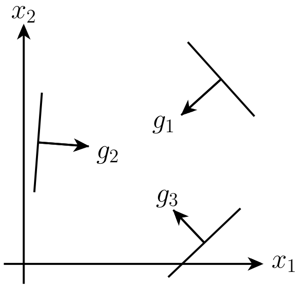
Important point: note that a maximum will always occur at a "corner" of the polytope (this is exactly the property that the simplex algorithm will exploit)

Unbounded and infeasible problems

Note that a linear program could be set up such that we could obtain infinite objective, if the polytope is unbounded in a direction of increasing c



Alternatively, polytope could be such that there are *no* feasible points (infeasible problem)



Example: suppose we have both the constraints $x_1 \geq 5$ and $x_1 \leq 4$

We cannot find a solution for unbounded or infeasible problems, but an algorithm should tell us that we are in either of these situations

Standard form linear programs

As a final step toward the simplex algorithm for solving linear programs, we consider linear programs in a slightly different form

Standard form linear program

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

where $x \in \mathbb{R}^n$ is optimization variable, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are problem data (m and n are new dimensions unrelated to the problem dimensions in inequality form); also a few technical conditions, like A should have full row rank, that we won't dwell on

Although they look different, easy to transform between inequality form and standard form

Converting to standard form

Simple steps to convert to standard form (presenting intuition here):

1. For any free variables x_i (variables not already constrained to be positive), add two variables indicating the positive and negative parts: x_i^+ and x_i^-; augment constraints to be
$$g_i^T x^+ - g_i^T x^- \leq h_i$$
(and similarly for objective terms c)

2. Add any equality constraints constraints in G (constraints with $g_i^T x \leq h_i$ and $-g_i^T x \leq h_i$) directly into A

3. For all inequality constraints, $g_i^T x \leq h_i$, add a new variable s_i, called a *slack variable*, and introduce the constraints
$$g_i^T x + s_i = h_i, \quad s_i \geq 0$$

4. Transform maximizing $c^T x$ to be minimizing $(-c)^T x$

Standard form example

Beginning with previous example:

$$\begin{aligned} \underset{x_1, x_2}{\text{maximize}} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 6 \\ & 3x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Variables are already non-negative so no need to introduce positive and negative parts

Introduce slack variables x_3, x_4 and constraints

$$\begin{aligned} x_1 + 2x_2 + x_3 &= 6 \\ 3x_1 + x_2 + x_4 &= 9 \\ x_3, x_4 &\geq 0 \end{aligned}$$

Our example problem is now in standard form:

$$\begin{array}{ll} \underset{x_1,x_2}{\text{maximize}} & 2x_1 + x_2 \\ \text{subject to} & x_1 + 2x_2 \leq 6 \\ & 3x_1 + x_2 \leq 9 \\ & x_1, x_2 \geq 0 \end{array} \quad \equiv \quad \begin{array}{ll} \underset{x}{\text{minimize}} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

with

$$c = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 0 \end{bmatrix}, \; A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Finding “corners” in standard form polytope

In standard form we assume $n > m$, so $Ax = b$ is an underdetermined system of equations (more variables than equations)

We can find solutions by selecting a *subset* of m columns of A, solving the linear system, and setting remaining $n - m$ variables to be 0

Those solutions that also satisfy $x \geq 0$ are the corners of the polytope

Aside: set notation for vectors/matrices

Some set notation will make the resulting systems easier to write

We let $\mathcal{I} \subseteq \{1, \ldots, n\}$ denote an *ordered index set*, and let $\mathcal{I}_i$ the denote the ith element in set

$$\mathcal{I} = \{\mathcal{I}_1, \mathcal{I}_2, \ldots \mathcal{I}_m\}$$

where $|\mathcal{I}| = m$ denotes the length of the index set

Unlike typical sets order is relevant, so that $\mathcal{I} = \{1, 5\}$ and $\mathcal{I} = \{5, 1\}$ are distinguished between (not important for basic algorithm, but it will be important when we talk about faster versions)

For a vector $x \in \mathbb{R}^n$ and index set $\mathcal{I}$, $x_{\mathcal{I}} \in \mathbb{R}^m$ is a vector of entries selected by indices in $\mathcal{I}$

$$x_{\mathcal{I}} = \begin{bmatrix} x_{\mathcal{I}_1} \\ x_{\mathcal{I}_2} \\ \vdots \\ x_{\mathcal{I}_m} \end{bmatrix}$$

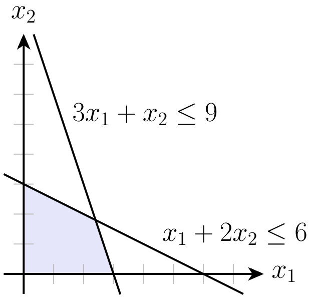
For a matrix $A \in \mathbb{R}^{m \times n}$, $A_{\mathcal{I}} \in \mathbb{R}^{m \times m}$ is a matrix where *columns* are selected by $\mathcal{I}$

$$A = \begin{bmatrix} | & | & & | \\ a_{\mathcal{I}_1} & a_{\mathcal{I}_2} & \cdots & a_{\mathcal{I}_m} \\ | & | & & | \end{bmatrix}$$

Example: polytope corners

Standard form polytope:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \ b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$



Example: polytope corners

Standard form polytope:

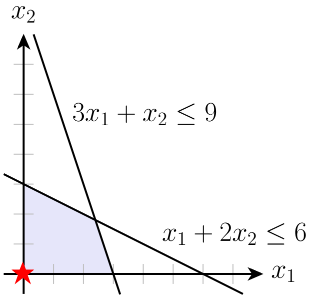
$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \ b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Choose index $I = \{3, 4\}$:

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

$$\Longrightarrow x = \begin{bmatrix} 0 \\ 0 \\ 6 \\ 9 \end{bmatrix}$$



Example: polytope corners

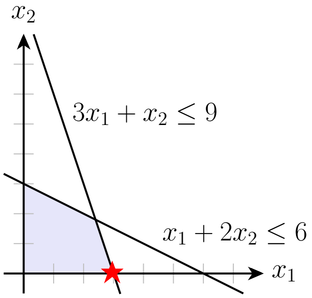
Standard form polytope:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \ b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Choose index $I = \{1, 3\}$:

$$\begin{aligned} x_{\mathcal{I}} &= A_{\mathcal{I}}^{-1} b \\ &= \begin{bmatrix} 1 & 1 \\ 3 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \end{aligned}$$

$$\Longrightarrow x = \begin{bmatrix} 3 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$



Example: polytope corners

Standard form polytope:

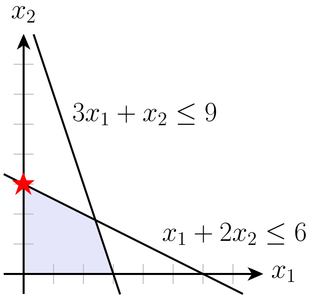
$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \ b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Choose index $I = \{2, 4\}$:

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b$$

$$= \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

$$\Longrightarrow x = \begin{bmatrix} 0 \\ 3 \\ 0 \\ 6 \end{bmatrix}$$



Example: polytope corners

Standard form polytope:

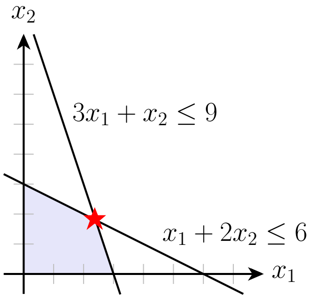
$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Choose index $I = \{1, 2\}$:

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b$$

$$= \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 2.4 \\ 1.8 \end{bmatrix}$$

$$\Longrightarrow x = \begin{bmatrix} 2.4 \\ 1.8 \\ 0 \\ 0 \end{bmatrix}$$



Example: polytope corners

Standard form polytope:

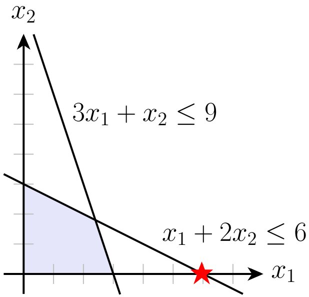
$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

Choose index $I = \{1, 4\}$:

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b$$

$$= \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 6 \\ -9 \end{bmatrix}$$

$$\implies x = \begin{bmatrix} 6 \\ 0 \\ 0 \\ -9 \end{bmatrix}$$



Naive linear programming algorithm

For each length-m subject $\mathcal{I} \subseteq \{1, \ldots, n\}$:

1. Compute $x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b$, and set $x_j = 0$ for $j \notin \mathcal{I}$

2. If $x \geq 0$, compute objective $c^T x$

Return x with lowest computed objective

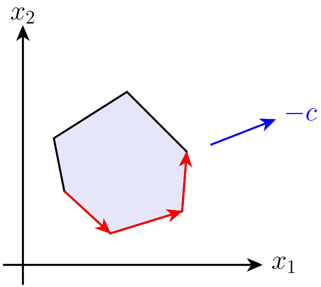
But, this requires testing $\binom{n}{m} = O(n^m)$ different subsets $\mathcal{I}$

The simplex method is a way of speeding up this “corner finding” process substantially

Outline

Simplex algorithm

Basic idea of simplex algorithm: move along edges of polytope from corner to corner, in directions of decreasing cost



Exponential complexity in the worst case (need to check a number of corners that is exponential in $n - m$), but it has notable advantages over even the best polynomial time linear programming algorithms (interior point primal-dual methods)

A single step of simplex algorithm

Consider again problem in standard form

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

Suppose we are at a feasible initial point; that is, we have some index subset $\mathcal{I}$ such that

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b \geq 0$$

(seems like a big assumption, we'll relax this shortly)

Now suppose that we want to increase x_j for some $j \notin \mathcal{I}$ (so we must have $x_j = 0$ initially)

We cannot just add some amount to x_j, e.g. $x_j \leftarrow \alpha$ for $\alpha > 0,$ because the resulting vector would not satisfy $Ax = b$

$$A_{\mathcal{I}} x_{\mathcal{I}} = b \implies A_{\mathcal{I}} x_{\mathcal{I}} + \alpha a_j \neq b$$

Instead, we need to *also* adjust $x_{\mathcal{I}}$ by some $d_{\mathcal{I}}$ to guarantee the resulting equation still holds

$$\begin{aligned} & A_{\mathcal{I}}(x_{\mathcal{I}} + \alpha d_{\mathcal{I}}) + \alpha a_j = b \\ \implies & \alpha A_{\mathcal{I}} d_{\mathcal{I}} = b - A_{\mathcal{I}} x_{\mathcal{I}} - \alpha a_j \\ \implies & d_{\mathcal{I}} = -A_{\mathcal{I}}^{-1} a_j \end{aligned}$$

Now, supposing we do adjust x by: $x_j \leftarrow \alpha$, $x_{\mathcal{I}} \leftarrow x_{\mathcal{I}} + \alpha d_{\mathcal{I}}$, how does this affect the objective function $c^T x$?

$$c^T x \leftarrow c^T x + \alpha(c_j + c_{\mathcal{I}}^T d_{\mathcal{I}})$$

In other words, adding α to x_j increases objective by $\alpha \bar{c}_j$ where

$$\bar{c}_j = c_j - c_{\mathcal{I}}^T A_{\mathcal{I}}^{-1} a_j$$

Thus, as long as $\bar{c}_j$ is negative (remember, we're trying to minimize $c^T x$), it's a "good idea" to adjust x in this manner

May be multiple js that decrease objective, turns out just trying them in order and picking the first with negative $\bar{c}_j$ works well (more on this later)

If no $\bar{c}_j < 0$, no improvement direction: we have found solution!

Final question: how big of a step $x_j \leftarrow \alpha$ should we take?

If all $d_{\mathcal{I}} \geq 0$ (and $\bar{c}_j < 0$), we are in “luck”, we can decrease the objective arbitrarily without ever leaving the feasible set $x \geq 0$, $\Longrightarrow$ *unbounded problem*

Otherwise, if some $d_{\mathcal{I}_i}$ is negative, we can only as big a step such that $x_{\mathcal{I}_i} + \alpha d_{\mathcal{I}_i} \geq 0$

So, let’s take as big as step as we can, until the first $x_{i^\star}$ hits zero, where

$$i^\star = \underset{i \in \mathcal{I} : d_i < 0}{\operatorname{argmin}} -x_i / d_i$$

At this point, element $i^\star$ *leaves* the index set $\mathcal{I}$, and j enters the set

Summary of simplex method

Repeat:

1. Given index set $\mathcal{I}$ such that $x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b \geq 0$

2. Find j for which $\bar{c}_j = c_j - c_{\mathcal{I}}^T A_{\mathcal{I}}^{-1} a_j < 0$ (if non exists, return $x_{\mathcal{I}}^\star = A_{\mathcal{I}}^{-1} b$)

3. Compute step direction $d_{\mathcal{I}} = -A_{\mathcal{I}}^{-1} a_j$ and determine index to remove (or return unbounded if $d \geq 0$)

$$i^\star = \underset{i \in \mathcal{I}: d_i < 0}{\operatorname{argmin}} -x_i / d_i$$

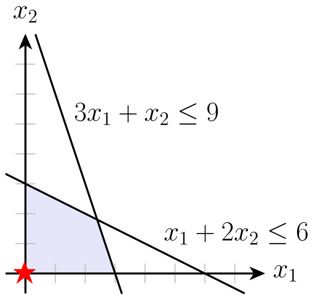
4. Update index set

$$\mathcal{I} \leftarrow \mathcal{I} - \{i^\star\} \cup \{j\}$$

Illustration of simplex

Standard form problem:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \; c = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$



$\mathcal{I} = \{3, 4\}$:

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

$$\bar{c}_{1,2} = (-2, -1)$$

Choosing $j = 1$:

$$d_{\mathcal{I}} = -A^{-1} a_1 = \begin{bmatrix} -1 \\ -3 \end{bmatrix}$$

$$i^{\star} = \underset{i \in \{3,4\}}{\operatorname{argmin}} \{3 : 6/1, 4 : 9/3\} = 4$$

$$\mathcal{I} \leftarrow \mathcal{I} - \{4\} \cup \{1\}$$

Illustration of simplex

Standard form problem:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \; c = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

$\mathcal{I} = \{1, 3\}$:

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\bar{c}_{2,4} = (-2/3, 1/3)$$

Choosing $j = 2$:

$$d_{\mathcal{I}} = -A^{-1} a_2 = \begin{bmatrix} -1/3 \\ -5/3 \end{bmatrix}$$

$$i^\star = \operatorname*{argmin}_{i \in \{1,3\}} \{1 : 3/(1/3), 3 : 9/(5/3)\} = 3$$

$$\mathcal{I} \leftarrow \mathcal{I} - \{3\} \cup \{2\}$$

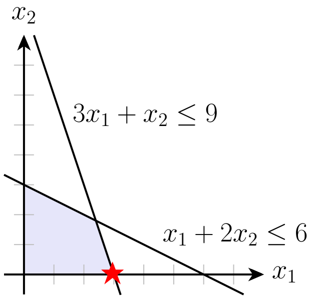
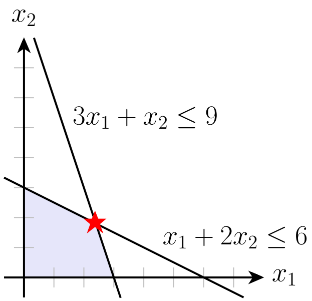


Illustration of simplex

Standard form problem:

$$A = \begin{bmatrix} 1 & 2 & 1 & 0 \\ 3 & 1 & 0 & 1 \end{bmatrix}, \; b = \begin{bmatrix} 6 \\ 9 \end{bmatrix}, \; c = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$



$\mathcal{I} = \{1, 2\}$:

$$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b = \begin{bmatrix} 2.4 \\ 1.8 \end{bmatrix}$$

$$\bar{c}_{3,4} = (0.2, 0.6)$$

Since all $\bar{c}_j$ are positive, we are at optimal solution

Numerical considerations

Note that the above algorithm is described in terms of exact math

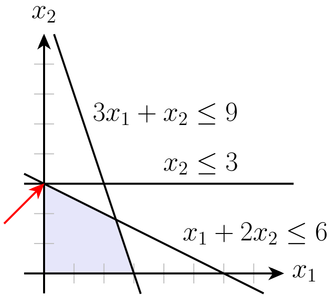
When implemented in floating point arithmetic, we'll often get entries like $\bar{c}_j, d_i \in [-10^{-15}, 10^{-15}]$

When comparing to zero, or comparing "ties" (especially under degeneracy, see next slide), we need to account for this

In practice, set these near-zero elements to zero or compare $\geq, \leq$ to $\pm 10^{-12}$

Degeneracy

More than m constraints may intersect at a given point, at such a corner the simplex solution will $x_i = 0$ for some $i \in \mathcal{I}$



To "make progress" simplex algorithm may have to take a step with $\alpha = 0$ (i.e., remain at the same point, but switch which column is in the index set)

Handling degeneracy

Simplex will still work fine with zero step sizes, but care must be taken to prevent *cycling* repeatedly over the same indices

A similar issue can occur when determining which $i^\star$ exits the index set (more than one x_i becomes zero at the same time)

A simple approach to fix these issues, *Bland's rule*:

1. At each step, choose smallest j such that $\bar{c}_j < 0$

2. From variables x_i that could exit index set, choose smallest i

Alternatively, perturb b by some small noise and then solve

Finding feasible solutions

We assumed a feasible initial point (i.e. $\mathcal{I}$ such that $A_{\mathcal{I}} b \geq 0$)

To find such a point (in cases where it is not easy to construct one), we introduce variables $y \in \mathbb{R}^m$ and solve auxiliary problem

$$\begin{array}{ll} \underset{x,y}{\text{minimize}} & 1^T y \\ \text{subject to} & Ax + y = b \\ & x, y \geq 0 \end{array} \tag{1}$$

A standard form linear program, with feasible solution $x = 0$, $y = b$, (if $b_i < 0$, replace constraint $a_i^T x = b_i$ with $-a_i^T x = -b_i$)

If solution to this problem has $y = 0$, we know $Ax = b$, so we have found a feasible solution to the original problem (though need to handle possible degeneracy)

Revised simplex

Simplex algorithm requires inverting $A_{\mathcal{I}}$ at each iteration: naive implementation would re-invert this matrix every time $\Longrightarrow O(m^3)$ computation at each iteration

Revised simplex algorithm directly maintains and updates this inverse $\mathcal{A}_I^{-1}$ at each iteration, using $O(m^2)$ computation

Key idea is the *Sherman-Morrison* inversion formula, for an invertible matrix $C \in \mathbb{R}^{n \times n}$ and vectors $u, v \in \mathbb{R}^n$

$$(C + uv^T)^{-1} = C^{-1} - \frac{C^{-1}uv^TC^{-1}}{1 + v^TC^{-1}u}$$

Suppose we update $\mathcal{I}$ by dropping index $\mathcal{I}_k$ and adding index j

Numerically, this is done by overwriting the column $a_{\mathcal{I}_k}$ in $A_{\mathcal{I}}$ with a_j (suppose this occurs at index I_k), i.e.,

$$A_{\mathcal{I}} \leftarrow A_{\mathcal{I}} + (a_j - a_{\mathcal{I}_k})e_k^T$$

where e_k is the kth basis vector (all zeros except for 1 at element k)

Using the Sherman-Morrison formula

$$(A_{\mathcal{I}} + (a_j - a_{\mathcal{I}_k})e_k^T)^{-1} = A_{\mathcal{I}}^{-1} - \frac{A_{\mathcal{I}}^{-1}(a_j - a_{\mathcal{I}_k})e_k^T A_{\mathcal{I}}^{-1}}{1 + e_k^T A_{\mathcal{I}}^{-1}(a_j - a_{\mathcal{I}_k})}$$

Right most expression can be written as an outer product matrix uv^T, so forming and adding it to $A_{\mathcal{I}}^{-1}$ takes $O(m^2)$ operations

Simplex tableau

If you've been taught the simplex algorithm before (or if you read virtually any book on the subject), you have probably see tables that look like this

	6.6	0	0	0.2	0.6
1	2.4	1	0	-0.2	0.4
2	1.8	0	1	0.6	-0.2

This is the *simplex tableau*, and it is simply an organization of all the relevant quantities of the simplex algorithm in a table

	$-c^T x$	$\bar{c}^T = c^T - c_{\mathcal{I}}^T A_{\mathcal{I}}^{-1} A$
$\mathcal{I}$	$x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b$	$A_{\mathcal{I}}^{-1} A$

plus a set of operators for performing the updates (essentially doing the same thing as the Sherman-Morrison formula)

Outline

Introduction

Some linear algebra review

Linear programming

Simplex algorithm

Duality and dual simplex

Lagrangian duality for LPs

Duality is an extremely powerful concept in convex optimization in general

Given a linear program in standard form

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

we define a function the *Lagrangian*, which takes the form

$$\mathcal{L}(x, y, z) = c^T x + y^T (Ax - b) - z^T x$$

where y and z are called *dual variables* for the constraints $Ax = b$ and $x \geq 0$

First note that

$$\max_{y,z\geq 0} \mathcal{L}(x,y,z) = \begin{cases} c^T x & \text{if } Ax = b \text{ and } x \geq 0 \\ \infty & \text{otherwise} \end{cases}$$

Thus we can write the original problem as

$$\min_{x} \max_{y,z\geq 0} \mathcal{L}(x,y,z)$$

where we are effectively using the min/max setup to express the same constraints as in the standard form problem

Alternatively, we can flip the order of the min/max to obtain what is called the *dual* problem

$$\max_{y,z\geq 0} \min_{x} \mathcal{L}(x,y,z)$$

Denoting $p^\star$ and $d^\star$ as the objectives for the primal and dual problems respectively, it is immediately clear that

$$d^\star \leq p^\star$$

(if we minimize over x first and then maximize over $y, z \geq 0$, this will always give a bigger objective than maximizing over $y, z \geq 0$ first and then minimizing over x), called *weak duality*

A remarkable property: for linear programs, we actually have $d^\star = p^\star$ (strong duality), and the simplex algorithm gives us solutions for *both* the primal and dual problems

Dual problem for standard form LPs

First note that the inner minimization in the dual problem is given by

$$\begin{aligned}\min_x L(x,y,z) &= \min_x c^T x + y^T(Ax-b) - z^T x \\ &= \begin{cases} -b^T y & \text{if } c + A^T y - z = 0 \\ -\infty & \text{otherwise} \end{cases}\end{aligned}$$

Thus we can write the dual problem as

$$\begin{array}{ll} \underset{y,z}{\text{maximize}} & -b^T y \\ \text{subject to} & A^T y + c = z \\ & z \geq 0 \end{array} \equiv \begin{array}{ll} \underset{y}{\text{maximize}} & -b^T y \\ \text{subject to} & -A^T y \leq c \end{array}$$

(a linear program in inequality form!)

Strong duality for LPs

For linear programs, strong duality holds ($p^\star = d^\star$) and simplex algorithm gives solutions $x^\star$ and $y^\star$

Proof: when simplex algorithm terminates, we have $x_I^\star = A_\mathcal{I}^{-1} b \geq 0$ and $c - A^T A_\mathcal{I}^{-T} c_\mathcal{I} \geq 0$ (no direction of cost decrease).
Letting $y = -A_\mathcal{I}^{-T} c_\mathcal{I}$ be a potential assignment to the dual variables, by the above condition we have

$$c + A^T y \geq 0, \quad \text{(i.e., } y \text{ is dual feasible)}$$

and

$$p^\star = c^T x^\star = c_\mathcal{I}^T A_\mathcal{I}^{-1} b = -b^T y$$

i.e., y is an optimal dual solution and $d^\star = p^\star$. $\square$

Dual simplex algorithm

In light of the above discussion, simplex algorithm can be viewed in the following manner:

1. At all steps, maintain primal feasible solution $x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b \geq 0$

2. Work to obtain dual feasible solution $y = -A_{\mathcal{I}}^{-T} c_{\mathcal{I}}$ such that $-A^T y \leq c$

There is an alternative approach, called the dual simplex algorithm, that works in the opposite manner

1. At all steps, maintain dual feasible solution $y = -A_{\mathcal{I}}^{-T} c_{\mathcal{I}}$ such that $-A^T y \leq c$

2. Work to obtain primal feasible solution $x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b \geq 0$

Dual simplex algorithm takes very similar form to the standard simplex (just showing algorithm here, not derivation, but it is similar to before)

Repeat:

1. Given index set $\mathcal{I}$ such that $y_{\mathcal{I}} = -A_{\mathcal{I}}^{-T} c_{\mathcal{I}}$ such that $-A^T y \leq c$

2. Letting $x_{\mathcal{I}} = A_{\mathcal{I}}^{-1} b$, find some $\mathcal{I}_k$ such that $x_{\mathcal{I}_k} \leq 0$

3. Let $v = A^T (A_{\mathcal{I}}^{-T})_k$ (where $(A_{\mathcal{I}}^{-T})_k$ denotes kth column of $A_{\mathcal{I}}^{-T}$) and determine index to add (or return infeasible if $v \geq 0$)

$$j = \operatorname*{argmin}_{j \notin \mathcal{I} : v_i < 0} -\frac{\bar{c}_i}{v_i}$$

4. Update index set

$$\mathcal{I} \leftarrow \mathcal{I} - \{\mathcal{I}_k\} \cup \{j\}$$

Incrementally adding constraints

In general, little reason to prefer dual simplex over standard simplex

Advantage comes when we can easily find an initial dual feasible solution, but not an initial primal feasible solution

Common scenario: adding an additional inequality constraint to a linear program

Suppose we have solved the standard form LP

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

and found primal and dual solutions $x^\star$ and $y^\star$

Now suppose we want to add the constraint

$$g^T x \leq h$$

Introducing slack variable x_{n+1} (with $c_{n+1} = 0$), this would make the linear equalities in standard form

$$\begin{bmatrix} A & 0 \\ g^T & 1 \end{bmatrix} \begin{bmatrix} x \\ x_{n+1} \end{bmatrix} = \begin{bmatrix} b \\ h \end{bmatrix}$$

Not trivial to modify $x^\star$ variables to attain primal feasibility

But, it is easy to see that $\begin{bmatrix} y^\star \\ 0 \end{bmatrix}$ is an *dual feasible* point

$$\begin{bmatrix} A^T & g \\ 0 & 1 \end{bmatrix} \begin{bmatrix} y^\star \\ 0 \end{bmatrix} \leq \begin{bmatrix} c \\ 0 \end{bmatrix}$$

Thus, we can initialize dual simplex with this solution, which often takes *much* less time than solving problem given no good initial solution

Will be especially important for integer programming, as solvers will constantly modify LPs by adding one additional constraint at a time