Fast Times in Linear Programming:
Early Success, Revolutions, and Mysteries

Margaret H. Wright

Computer Science Department

Courant Institute of Mathematical Sciences

New York University

Math Across Campus

University of Washington

October 21, 2011

Linear programming (LP) can be viewed as mathematics and/or computer science and/or operations research and/or economics.

LP is used, directly or indirectly, in business, science, and engineering.

And it makes a great story, much too long to tell in one lecture—so this talk will touch only on a few highlights.

Today's topics will include bits of mathematical algorithms, computational complexity, and scientific computing.

The obvious way to start: what is linear programming?

Historically, linear programming is a relatively modern field ($60+$ years old) of mathematics and computer science, dating from near the end of World War II (approximately 1945).

Note: "Programming" in those days (before modern computers) meant developing a plan, organization, or schedule.

Formally, linear programming is the problem of minimizing a linear objective function subject to linear constraints.

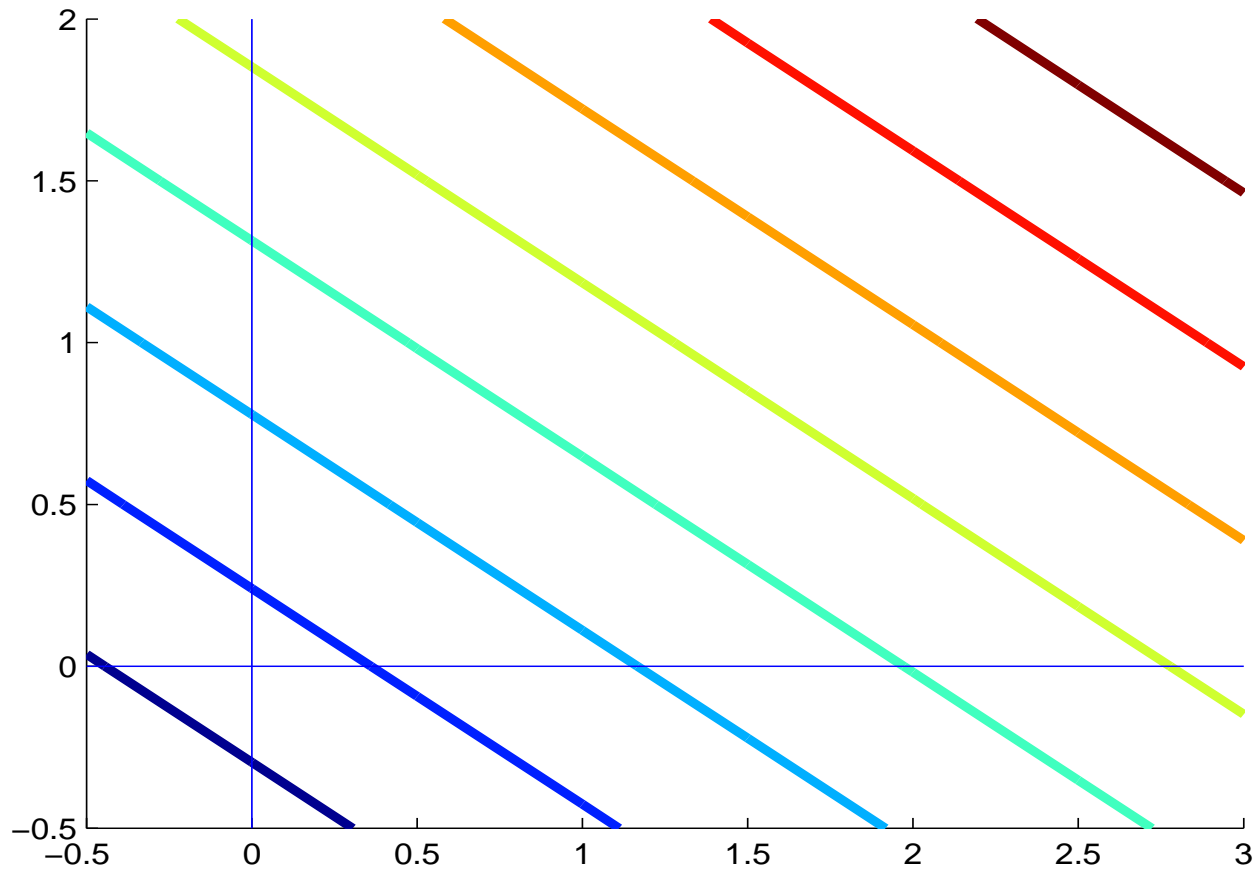Definition by example: a bad practice (like proof by intimidation)...

A linear function

$2x$ is a linear function of the variable x;

x^2 is a nonlinear function of x.

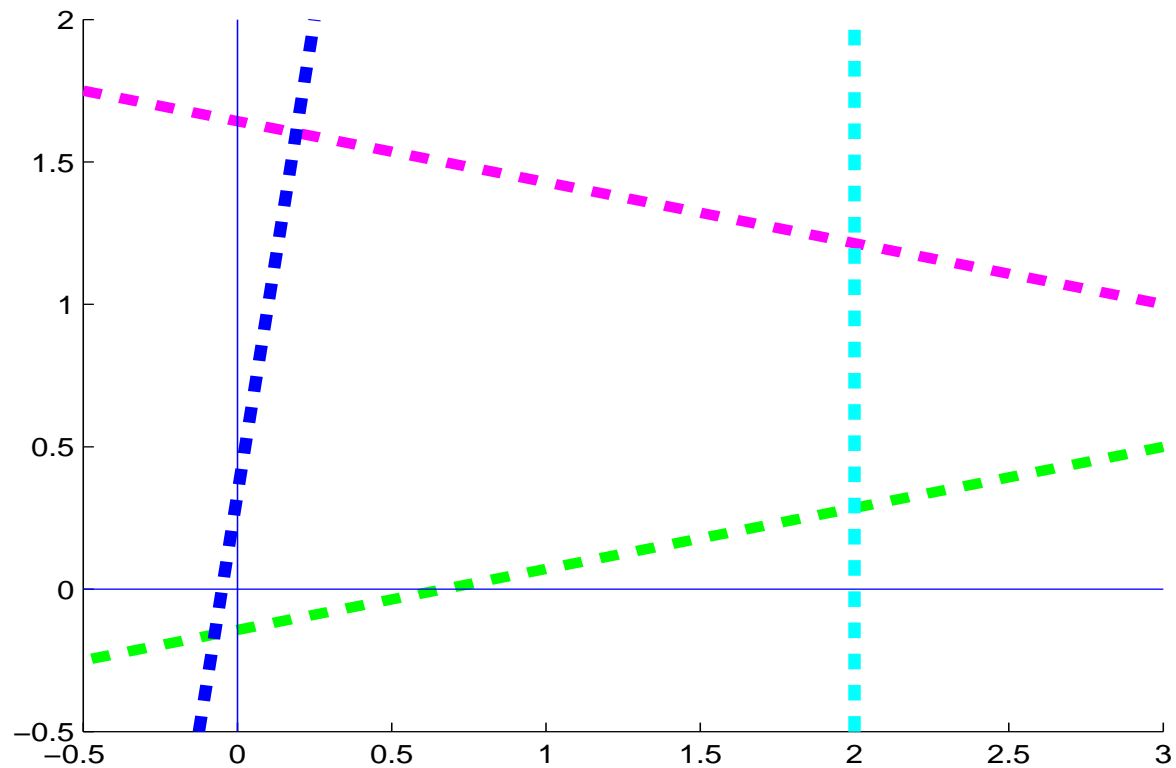$2x + 3y$ is a linear function of the pair (x, y);

$2x + y^2$ is a nonlinear function of the pair (x, y).

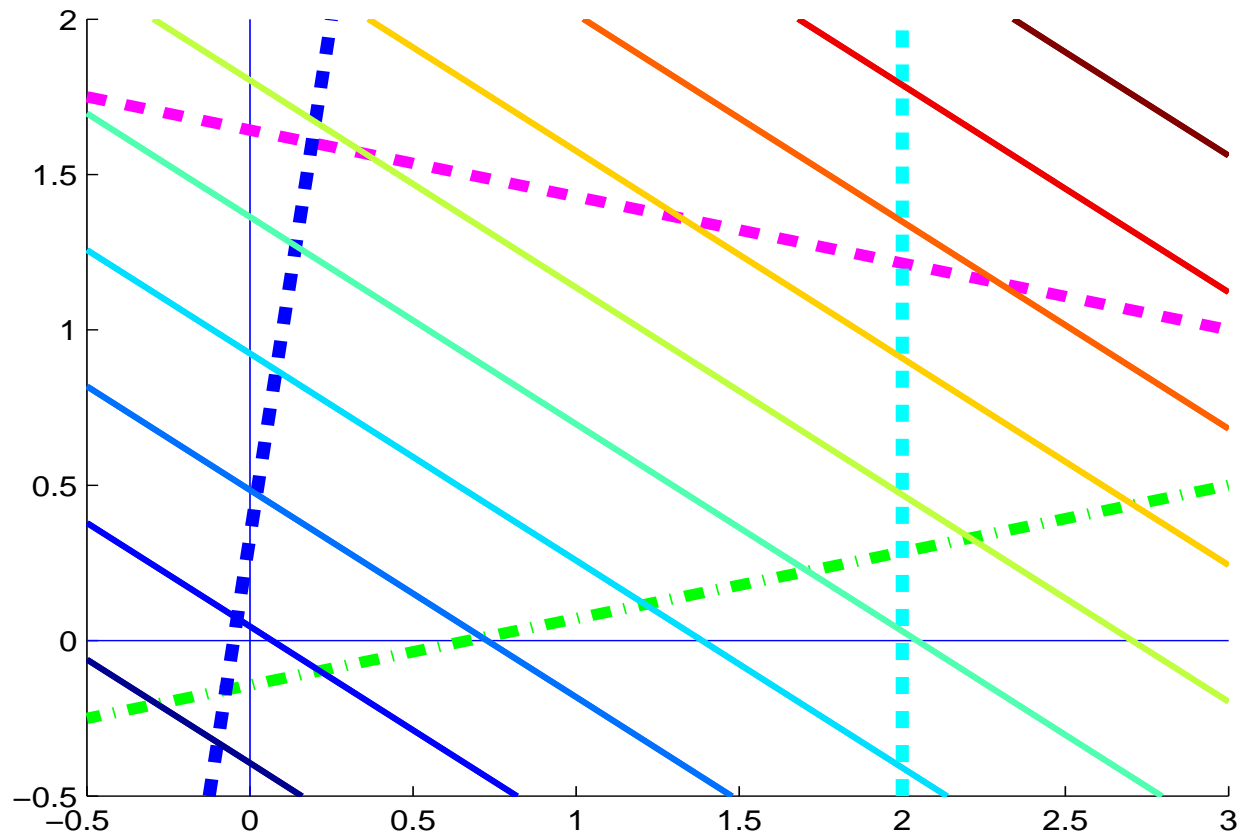In two dimensions, the contours of a linear function are straight lines.

"Linear constraints"?

An equality stays "on" one contour; an inequality stays *on or on one side of* a contour.

And "linear programming"?

One of the earliest applications of LP during and after World War II: feeding military troops an "optimal" diet.

For an everyday-life version, consider choosing an adequately nutritious breakfast that costs as little as possible.

Eight available items: pancakes; milk; freshly squeezed orange juice; scrambled eggs; Weetabix, a high-fiber cereal; fresh blueberries; bacon; and vitamin/mineral pills.

Our goal is to find the cheapest breakfast satisfying daily fiber ≥ 120, daily calcium ≥ 100, and all x's ≥ 0. (Can't eat negative units of a food.)

food	fiber	calcium	cost
pancakes	5	10	3
milk	0	40	1
orange juice	4	5	5
eggs	8	0	3
Weetabix	100	20	10
blueberries	50	20	5
bacon	0	0	10
pills	30	400	2

Is the solution obvious? Not for most people, although one might guess no bacon.

With this (not obviously crazy) formulation, what's the optimal breakfast??

1.14 units of Weetabix and .19 units of vitamin/mineral pills.

Yuck. Not even milk for the Weetabix!

What's wrong?

Not the mathematics. . . but rather the problem it was given to solve.

For one thing, not enough constraints were imposed, e.g., we need milk or OJ to go with Weetabix.

A point worth emphasizing (no time here, but we can discuss offline):

Mathematics can tell us a lot, but it should not be expected to figure out when the problem is wrong.

In the rest of this talk, I'll assume that the problems we're asked to solve are always the ones that people really want to solve.

LP problems sound simple—do they actually arise in anything important?

Oh yes!

If importance is measured in money, LP and its applications add up to billions of dollars per year.

In planning, scheduling, resource allocation, design of

- transportation

- communication

- finance and banking

- manufacturing

- energy

- health and agriculture

Measuring importance in scholarly terms, many areas of mathematics and computer science are inspired by and deeply connected with linear programming:

- convex geometry

- combinatorics

- properties of polytopes

- integer programming (combinatorial optimization)

- complexity theory

- machine learning

- compressed sensing (recovering sparse data)

How to solve a linear program? Insight by picture: the optimal solution (essentially always) is a *vertex*, i.e., a "corner point", so the number of possible solutions is finite.
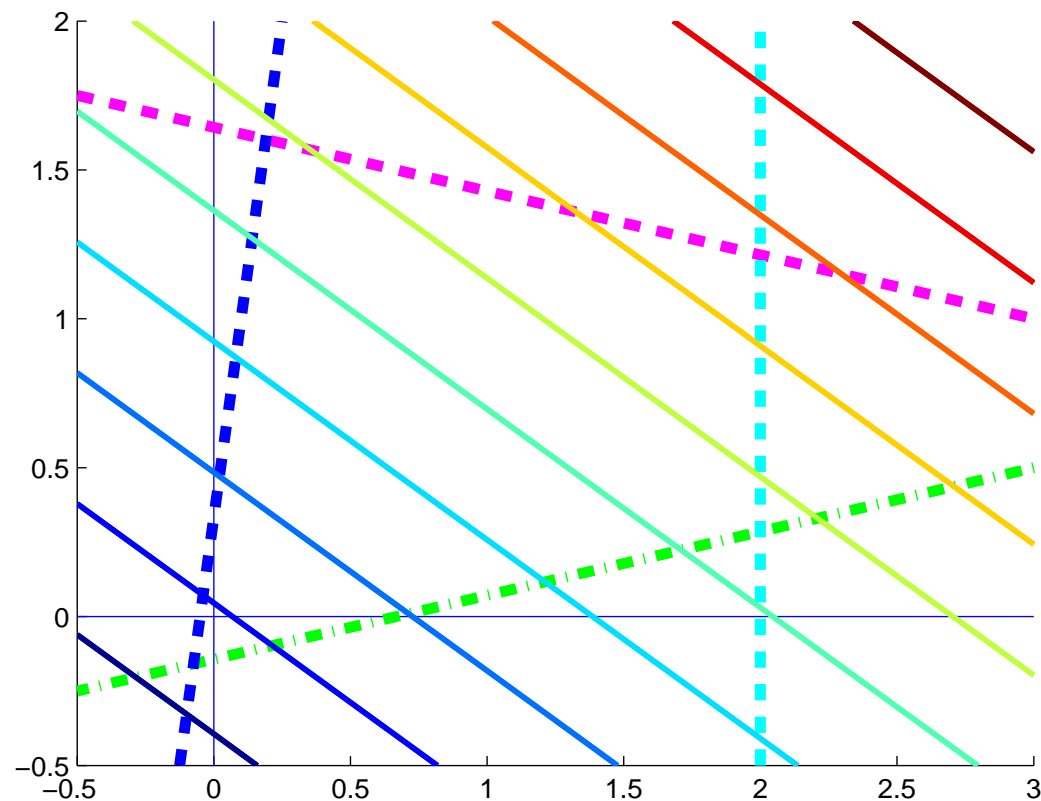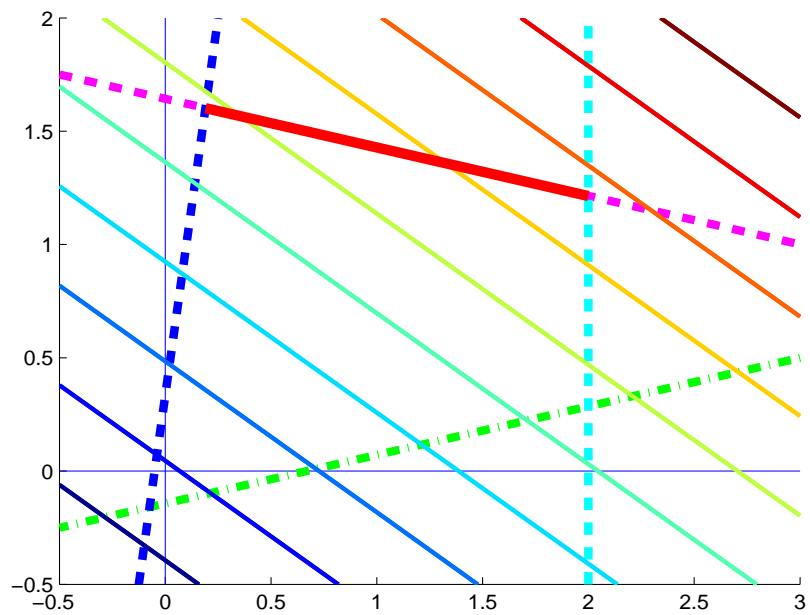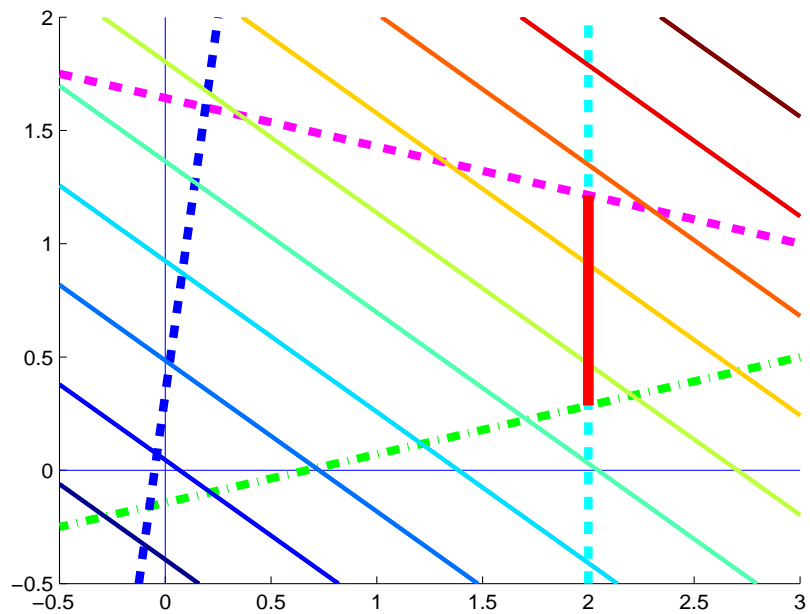
Based on this insight and others, just after World War II, George B. Dantzig (affectionately known as "GBD") invented the *simplex method*.

The idea is to start at a vertex and then move from vertex to vertex, reducing the objective function with every move.

The concept of the simplex method is extremely simple, but at the beginning no one, least of all GBD, thought it would work quickly enough to solve LPs of reasonable size. But it did, especially considering the available computing hardware.

In the late 1940s and early 1950s, decks of punched cards were run on calculators formed by connecting IBM office accounting equipment.

In 1950, a typical accounting machine had 80 mechanical counters, plus storage for 16 10-digit numbers relayed from an electronic calculator that could perform about 2,000 additions or subtractions per second, and 86 multiplications or divisions per second.

Running the Gauss-Seidel procedure inverted a 10×10 matrix in 30 minutes (!), with 7 correct digits.

Every simplex step requires solving a square nonsingular system (the "basis" or "working set") to determine whether the current vertex is optimal.

If it is not, moving to the next vertex ("pivoting") produces a matrix that differs from the previous one by a single row (or column).

Hence the linear systems do not need to be solved from scratch at each vertex—this is crucial to achieving fast times for the simplex method.

Many clever computational "tricks" (aka "techniques") were developed to do this. One of my favorites (1953):

"In the revised simplex method . . . each iteration replaces one of the columns of a matrix. In the product form of representation, this change can be conveniently effected by multiplying the previous matrix by an elementary matrix. Thus, only one additional column of information need be recorded with each iteration. . . . Using the IBM Card Programmed Calculator, a novel feature results: when the inverse matrix is needed at one stage and its transpose at another, this is achieved simply by turning over the deck of cards representing the inverse."

The simplex method worked so well that essentially no one thought about using any other method. Hence the mindset in linear programming was

1947   Simplex method (Dantzig)

1950s   Simplex method

1960s   Simplex method

1970s   Simplex method

:
:

Practitioners (i.e., people who solved real LPs) were very happy with the simplex method.

Why were they so happy?

Because the simplex method *essentially always* took $2n$–$3n$ steps to solve the problem, where n is the number of variables, and each step took $O(n^2)$ operations.

This property reminds us of something. . .

Computational complexity 1A:

A problem is in P if there is a polynomial-time algorithm to solve it, i.e., an algorithm whose work is bounded by a polynomial in a suitable measure of the problem's "size".

Since the mid-1960s, it's been close to an article of faith that a polynomial-time algorithm is inherently fast ("good").

And that an algorithm that might take exponential time is BAD.

What about the simplex method?

Although it behaved in practice like a polynomial-time algorithm, there was an obvious worry: it moves from vertex to vertex, and the number of vertices in an LP can be *exponential* in the problem dimension.

Hence the simplex method's great practical success did NOT convince the theoretical computer science community that it is a good algorithm.

A great (sometimes terribly annoying) thing: mathematicians and computer scientists are *really, really* good at finding nasty problems that cause algorithms to perform badly.

In 1972, Victor Klee (from the University of Washington) and George Minty produced an n-variable linear program whose (scaled) constraints form a "twisted hypercube" that has 2^n vertices, and for which the simplex method, if started at the "wrong" vertex, visits every one of those 2^n vertices.

Let $0 < \epsilon < \frac{1}{2}$.

$$
\begin{aligned}
\text{minimize} \quad & x_n \\
\text{subject to} \quad & 0 \leq x_1 \leq 1 \\
& \epsilon x_{j-1} \leq x_j \leq 1 - \epsilon x_{j-1}, \quad j = 2, \ldots, n.
\end{aligned}
$$

The "wrong" vertex is $(0, 0, \ldots, 0, 1)^T$.

Klee-Minty (scaled) in two dimensions ($2^2 = 4!$)
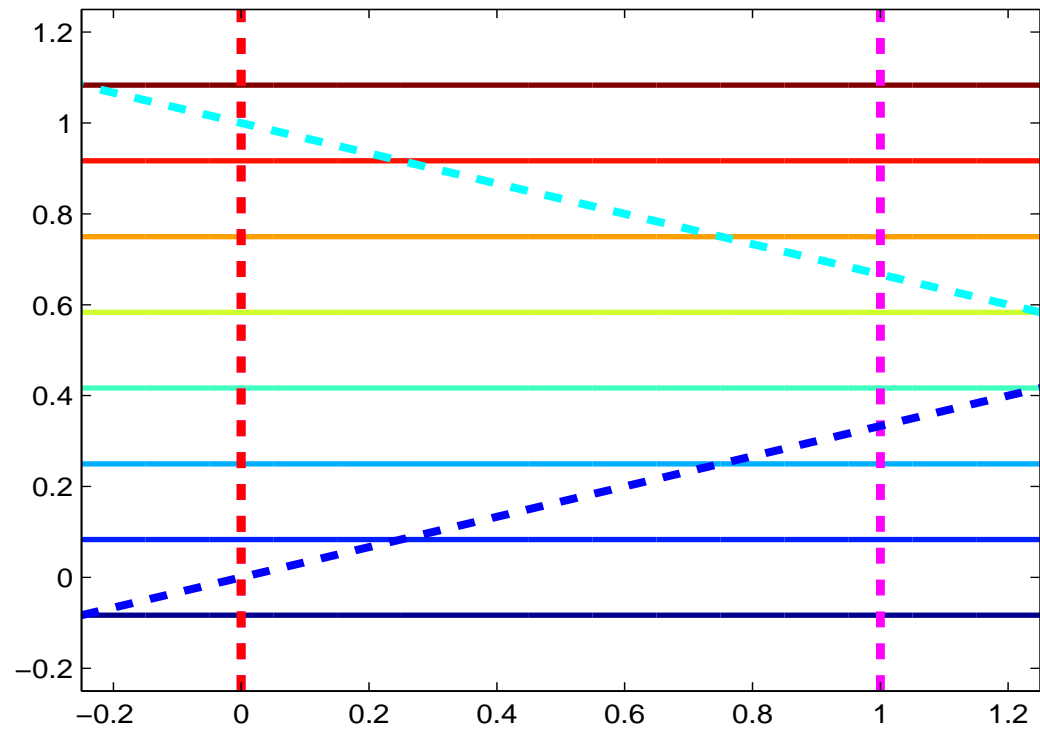
Result: in the worst case, with all known pivot rules ("pivoting" refers to deciding on the next vertex), the simplex method can take *exponential time*—which might seem to imply that the simplex method is a VERY bad algorithm.

But on the flip side of theory–practice, most practitioners who used LP tended not to worry about the possible worst case.

Even so, after Klee–Minty a nagging and logical question arose and remained open until 1979: linear programs seem "easy" to solve—is there a polynomial-time algorithm for LP?

The answer (after several false starts) came for most of us on November 7, 1979, in a headline on the front page of the *New York Times* (but below the fold):

A Soviet Discovery Rocks World of Mathematics

By MALCOLM W. BROWNE

A surprise discovery by an obscure Soviet mathematician has rocked the world of mathematics and computer analysis, and experts have begun exploring its practical applications.

Mathematicians describe the discovery by L.G. Khachian as a method by which computers can find guaranteed solutions to a class of very difficult problems that have hitherto been tackled on a kind of hit-or-miss basis.

Apart from its profound theoretical interest, the discovery may be applicable in weather prediction, complicated industrial processes, petroleum refining, the scheduling of workers at large factories, secret codes and many other things.

"I have been deluged with calls from virtually every department of government for an interpretation of the significance of this," a leading expert on computer methods, Dr. George B. Dantzig of Stanford University, said in an interview.

The solution of mathematical problems by computer must be broken down into a series of steps. One class of problem sometimes involves so many steps that it could take billions of years to compute.

The Russian discovery offers a way by which the number of steps in a solution can be dramatically reduced. It also offers the mathematician a way of learning quickly whether a problem has a solution or not, without having to complete the entire immense computation that may be required.

According to the American journal Sci-

The New York Times
Published: November 7, 1979
Copyright © The New York Times

A famous example of inaccurate science journalism:

"a method by which computers can find guaranteed solutions to a class of very difficult problems that have hitherto been tackled on a kind of hit-or-miss basis"

". . . a kind of problem related to the 'Traveling Salesman Problem', one of the most famous and intractable in mathematics"

"far-reaching consequences in such practical applications as the devising of secret codes"

My abstract mentioned international politics, and here it is... The Cold War was still going strong in 1979.

In the United States, the research managers for the Air Force, Army, Navy, and Department of Energy (all of whom had read the *New York Times* article) phoned the optimization researchers they supported to demand indignantly: "Why did the Soviets get this world-changing result and the Americans didn't?"

It turned out later that Khachiyan, who worked for the Soviet government, had been in trouble at his end for "giving this valuable information to the capitalists"!

All of us working on LP at that time tried in vain to explain to numerous science reporters what "polynomial-time" meant. (It's sad that these hilarious conversations were not recorded.)

Thus began the first revolution in linear programming.

For completeness, I need to mention that Khachiyan's ellipsoid method poses the LP problem in a different (but equivalent) way—as finding a point that satisfies a set of linear inequality constraints. Here's the idea:

- Define an initial ellipsoid known to enclose the feasible region;

- If its center is not feasible, construct a new (smaller) ellipsoid defined by a violated constraint;

- Keep going until a feasible point is found or the ellipsoid is small enough to confirm infeasibility.

The announcement of Khachiyan's result set off a frenzy around the world to implement his method and compare it with the simplex method.

Despite many algorithmic and numerical improvements, it became clear almost immediately that the ellipsoid method, in all variations, performed VERY poorly (in terms of solution time) compared to the simplex method.

So the ellipsoid algorithm turned out to be polynomial-time (a major theoretical breakthrough), but NOT FAST.

Looking back, what exactly was revolutionary?

The method was polynomial-time, but much more significantly (in my view)... Khachiyan's LP method was an adaptation of one for nonlinear convex programming proposed earlier by Yudin, Nemirovski, and Shor.

Using nonlinear techniques to solve linear programs ran completely contrary to the dominant "simplex" mindset and led to a change in perspective for all of optimization—a genuine (for once) paradigm shift.

Remember that, in 1984, linear programming and nonlinear programming were almost universally viewed as completely different.

The separation between linear and nonlinear programming was a fully accepted part of the culture of optimization.

So what had been happening all these years on the nonlinear side?

During the 1960s, major progress in unconstrained optimization had led researchers to think about ways to convert constrained problems into unconstrained problems.

Given an inequality-constrained problem

$$\underset{x \in \mathcal{R}^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad c_j(x) \geq 0, \quad j = 1, \ldots, m,$$

a popular solution method, based on the logarithmic barrier function, was to convert it to a sequence of unconstrained subproblems.

The idea is to create a composite function that combines $f(x)$ and a barrier term that has a singularity at the boundary (in effect, an infinitely high barrier preventing iterates from leaving the feasible region):

$$B(x, \mu) = f(x) - \mu \sum_{j=1}^{m} \ln c_j(x),$$

where $\mu > 0$ is called the *barrier parameter*.

During the 1960s, Fiacco and McCormick (and others) showed that the points $\{x_\mu\}$, the unconstrained minimizers of $B(x, \mu)$, define a smooth curve as $\mu \to 0$, called the *barrier trajectory* or the *central path*, that converges to the solution of the constrained problem from the strict interior of the feasible region—not along the boundary.

An obvious implied algorithm is to use Newton's method to find unconstrained minimizers of the log barrier function for a sequence of positive values of μ converging to zero.

Barrier (and closely related penalty) methods were widely used during the 1960s and early 1970s, but they were dogged by several concerns. The worry expressed most often in print involved the increasing ill-conditioning of the subproblems as $\mu \to 0$.

By 1984, methods seen as better had come along, and barrier methods were widely regarded as as old hat, unreliable, and to be deplored. Many courses and textbooks about nonlinear optimization did not even mention the existence of barrier methods.

Back to the LP side. . .

After Khachiyan, many theoretical computer scientists were upset and frustrated; the world was waiting for a polynomial-time algorithm that would beat the simplex method in speed.

And, in 1984, came the start of the second revolution (which, as it turns out, was also a renaissance):

On the front page of the *New York Times*, November 19, 1984, this time above the fold: an article headlined "Breakthrough in problem solving" described a polynomial-time algorithm for linear programming, invented by 28-year old Narendra Karmarkar of Bell Labs, claimed to be 50 times faster than the simplex method on every problem!

The tone was more cautious than the 1979 *Times* article about Khachiyan, but still somewhat breathless.

"In practice [the simplex method] usually manages to get there efficiently enough for most problems, as long as the number of variables is no more than 15,000 or 20,000. The Karmarkar algorithm, by contrast, takes a giant short cut, plunging through the middle of the solid"

In December 1984, *Time* magazine was less guarded (and less accurate) in "Folding the perfect corner", by Natalie Angier and Peter Stoler:

Some choice quotes:

"Narendra Karmarkar . . . after only a year's work has cracked the puzzle of linear programming by devising a new algorithm".

"Before the Karmarkar method, linear equations [sic] could be solved only in a cumbersome fashion, ironically known as the simplex method, devised by mathematician George Dantzig in 1947"

AT&T said that Karmarkar's algorithm was proprietary and could not be discussed in public (a major contrast to the situation 5 years earlier with Khachiyan). How could other researchers verify his claims?

After a sequence of events that I can describe offline, Karmarkar's algorithm was shown, in GMSTW (1985), to be formally equivalent to the log barrier method applied to LP. And, to the great surprise of many, in numerical tests on a standard LP test set, the primal log barrier method turned out to be competitive with the simplex method (although not 50 times faster on every problem).

The interior-point revolution was underway.

# Barrier/interior methods really are different from simplex!

With the all-inequality form of linear programming, in which we minimize $c^T x$ subject to $Ax \geq b$, the log barrier function is

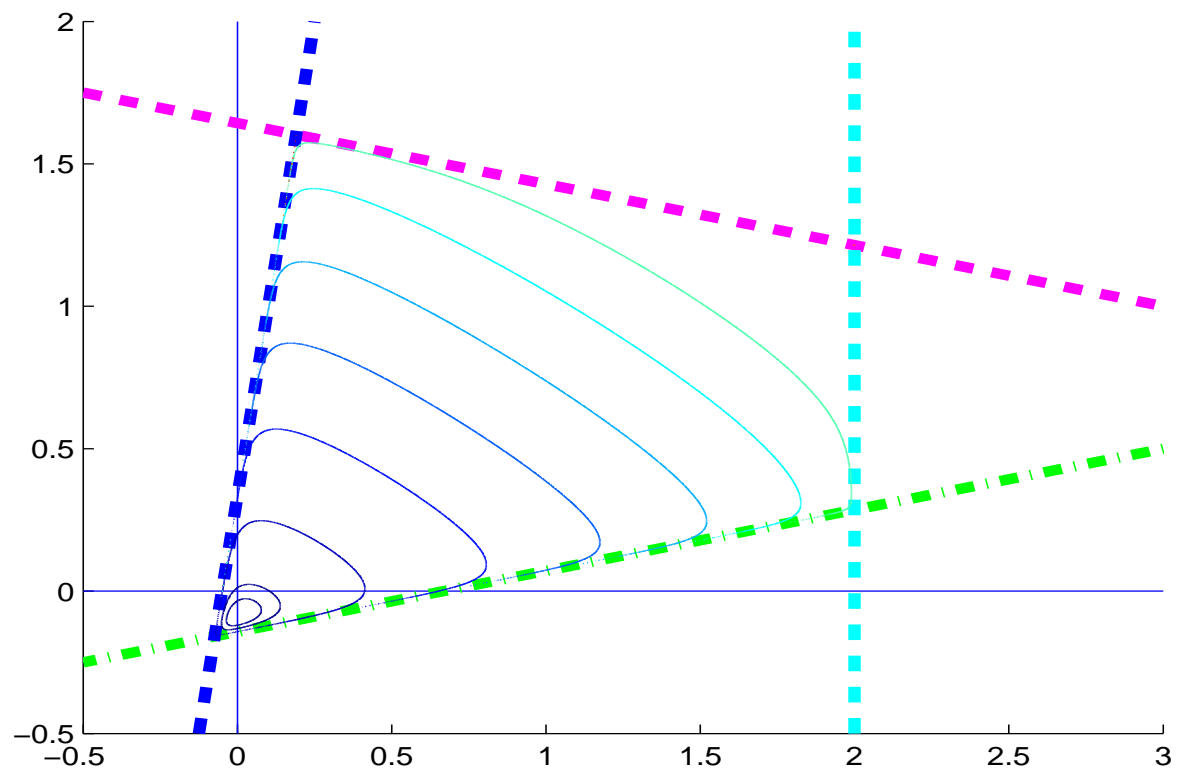$$B(x, \mu) = c^T x - \mu \sum_{i=1}^{m} \log(a_i^T x - b_i).$$

It's easy to see that the barrier function is very badly behaved near the boundaries of the feasible region, as shown next for $\mu = 5$ and $\mu = 0.1$.

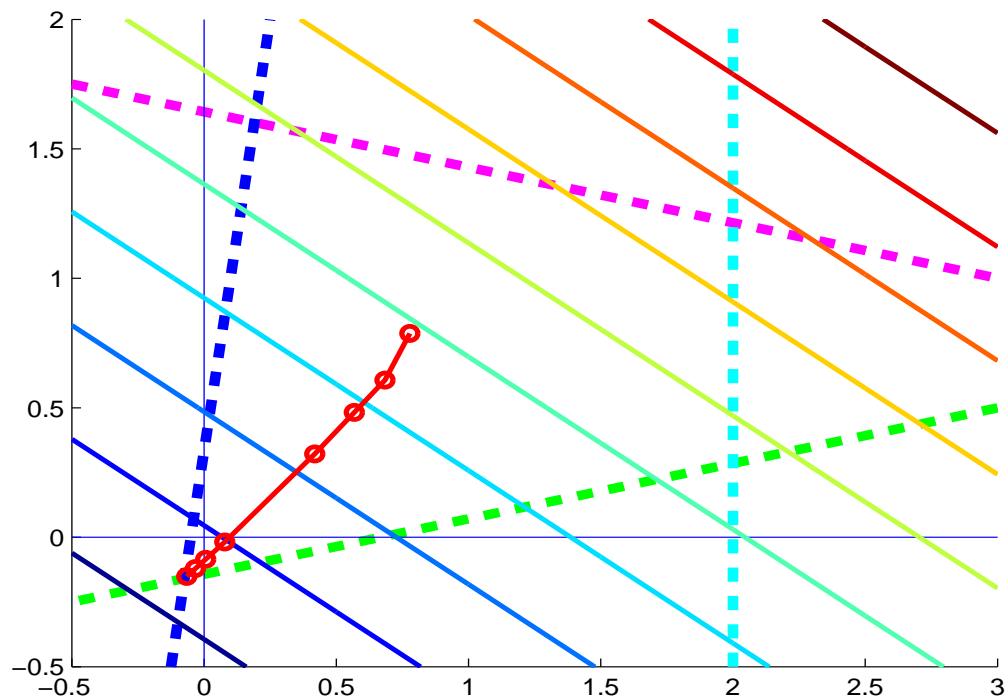(Both were challenging for Matlab's "contour" command.)

$\mu = 5$
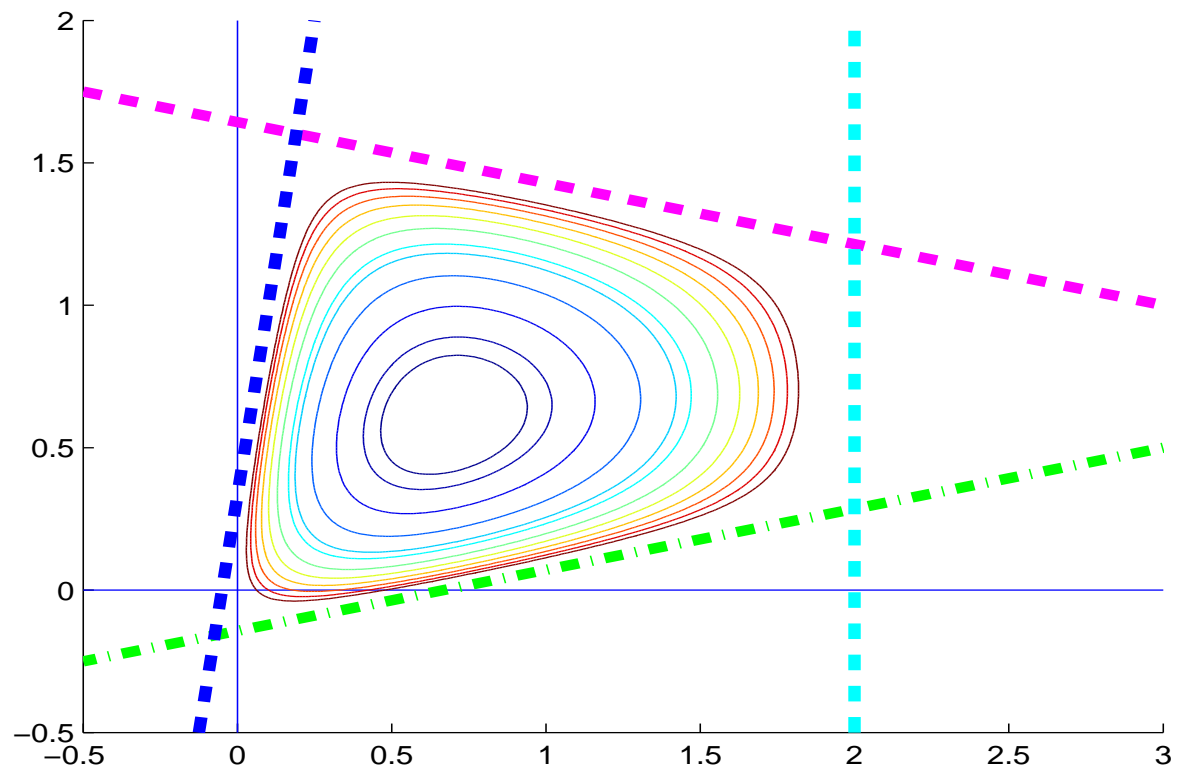
$\mu = 0.1$

Here's a typical path of minimizers as $\mu \to 0$, clearly approaching the solution from the interior of the constraints, not along the boundary.

And today? Decades have passed and many things have happened.

Devotees of the simplex method, stunned by the challenge to its dominance, have been motivated to improve it, in many ways. (But it's still the simplex method.)

Interior/barrier methods are used not just for linear programming, but for nonlinear optimization and other forms of convex optimization such as semidefinite programming.

For a very recent survey, see J. Gondzio, "Interior point methods 25 years later", *European Journal of Operational Research* (2011).

One happy development: new insights have been obtained about the gap between the worst-case exponential and the "in practice" polynomial-time performance of simplex.

In their justifiably famous paper "Smoothed analysis of algorithms: why the simplex method usually takes polynomial time" (2002), Spielman and Teng introduce smoothed analysis, a hybrid of worst-case and average-case analyses that attempts to circumvent the need for a theoretical definition of "practical inputs".

They then go on to show that the simplex algorithm "has smoothed complexity polynomial in the input size and the standard deviation of Gaussian perturbations".

The details of smoothed analysis are very complicated, but here's the concept.

Smoothed analysis examines a neighborhood of every input instance, and measures the worst case of the algorithm's expected performance when applied to *small random perturbations* of all possible inputs.

A favorable smoothed analysis means that an algorithm will perform well on small perturbations of all inputs. If the worst cases occur very seldom (think of Klee-Minty), and are very isolated, they will be ruled out by slight perturbations of the input.

Two interesting, not fully resolved, mysteries remain:

1. Why do both simplex and interior methods solve LPs much faster than they have any right to do?

2. Why are there differences in speed between interior methods and simplex?

The second question features in Bixby (2002) , which contains a wide range of numerical tests comparing different solution methods on very large LPs. (Bixby was the original developer of CPLEX, state-of-the-art LP software.)

680 LP models were tested, with m (number of constraints) up to 7 million, imposing a time limit of 4 days per LP, using version 8.0 of CPLEX. NB: these results are out of date, but the overall conclusions haven't changed.

m	Number of LPs tested
> 0	680
> 10000	248
> 100000	73

An especially interesting set of results compare primal and dual simplex with barrier.

Overall, even for the largest problems, the best simplex was approximately comparable with barrier—sometimes much better, sometimes much worse, sometimes similar.

Dual simplex is usually better than primal, but not always.

Perhaps surprisingly, it's very hard to make definitive statements about which algorithm will be the fastest.

And we still don't understand why the different methods perform very differently on problems that we believe to be "similar", or to have "similar structure".

Problem 1: approximately 5 million variables and 7 million constraints

Problem 2, "very similar in structure": 750,000 constraints

	Primal simplex	Dual simplex	Barrier
Problem 1	1880	6413	5642
Problem 2	1.8×10^6	48.3	28,161

????????????????

Summary:

On the bright side, algorithmic improvements in both simplex and interior-point methods, plus faster computers, mean that huge linear programs considered "challenging" or impossible 15 years ago can now be solved rapidly and routinely.

And. . . if we only knew when certain algorithms work well *and* could test whether these conditions hold, we could solve linear programs even faster—and we might be able to develop new approaches so that LP will be more effective in crucial applications, such as discrete optimization and compressed sensing.

Lots of future research!