# In Search of the Optimum

A History of Mathematical Optimization from Fermat to Karmarkar

Troy Altus

2025-12-31

# Table of contents

# The Shape of the Possible

**Learning Objectives**

- Understand what optimization means as a mathematical discipline
- Trace the isoperimetric problem from ancient Greece to modern engineering
- See how Fermat's method of stationary points planted the seed of calculus-based optimization
- Recognize optimization as a unifying thread across disparate fields

## Queen Dido's Problem

According to Virgil, when the Phoenician princess Dido arrived on the coast of North Africa around 900 BCE, she struck a bargain with the local king. She could have as much land as she could enclose with a single oxhide. Dido cut the hide into thin strips, joined them end to end, and used the resulting length of leather to enclose a semicircle against the sea. She got more land than the king intended to give.

Whether or not Dido existed, the problem she solved is real. Given a fixed length of rope, what shape encloses the maximum area? The answer — a circle — was known empirically by the ancient Greeks, suspected to be true for centuries, and not proved rigorously until 1838, when the Swiss mathematician Jakob Steiner published five different proofs, all beautiful, all flawed. The first complete proof came from Karl Weierstrass in 1870, using a technique that would not be fully developed until the twentieth century.

The isoperimetric problem — *equal perimeter, maximum area* — is the oldest optimization problem in mathematics. It has a clean answer, a compelling physical interpretation, and a proof that eluded some of the greatest mathematicians of the nineteenth century. It also captures, in a single sentence, what optimization is about: given a constraint, find the best.

## What Optimization Means

Optimization, as a mathematical discipline, asks a specific question: given a function $f(\mathbf{x})$ and a set of constraints, find the value of $\mathbf{x}$ that makes $f$ as small (or as large) as possible.

The function $f$ is called the *objective*. The constraints define the *feasible set* — the region of valid solutions. The solution, if it exists, is the *optimum*. These three elements — objective, feasible set, optimum — appear in every optimization problem from Dido's oxhide to a solid rocket motor grain design.

The simplicity of this formulation is deceptive. The objective might be smooth and convex, in which case the optimum is unique and easy to find. Or it might be riddled with local minima, discontinuous, noisy, or defined only implicitly through a simulation code. The feasible set might be a simple box, a complicated surface in high-dimensional space, or a discrete set of integers. The optimum might be well-defined and easy to characterize, or it might not exist at all.

The history of optimization is largely a history of mathematicians learning to handle these complications — extending the simple idea of finding a minimum to cover the full range of problems that the physical and economic world actually presents.

## Fermat and the Stationary Point

The first systematic technique for finding optima arrived in the seventeenth century, before calculus existed as a unified subject. Pierre de Fermat, the French lawyer and amateur mathematician who is better known for his work in number theory, noticed something about the maximum of a function.

Consider a parabola opening downward, reaching its peak at some point $x^*$. At the peak, the tangent is horizontal — the function neither rises nor falls. Fermat formalized this observation: at an interior maximum or minimum of a smooth function, the derivative is zero. This is Fermat's theorem on stationary points, and it is the foundation of every gradient-based optimization method that has ever been invented.

Fermat stated his result in 1638, in the language of algebraic geometry rather than calculus, but the idea is unmistakably modern. He wrote that at the point of maximum or minimum, the value of the function is "adequal" to its value at a nearby point — meaning the difference is negligible. Substitute $x + \epsilon$ for $x$, compute the expression, set the terms involving $\epsilon$ to zero, and solve. This is essentially the method of setting the derivative equal to zero, discovered before the derivative had a formal definition.

Isaac Newton and Gottfried Leibniz formalized Fermat's intuition into calculus in the 1660s and 1670s. The machinery of derivatives made Fermat's method precise: a necessary condition for an interior minimum or maximum of a differentiable function is that the derivative vanishes. The condition is necessary but not sufficient — it identifies candidates (stationary points) but does not distinguish minima from maxima or saddle points. Distinguishing these requires the second derivative.

## The Brachistochrone Problem and What It Started

In 1696, Johann Bernoulli posed the brachistochrone problem in the pages of the journal *Acta Eruditorum*. A bead slides without friction along a wire connecting two points. What shape should the wire take to minimize the time of descent?

The problem is phrased as a challenge: "I, Johann Bernoulli, address the most brilliant mathematicians in the world. Nothing is more attractive to intelligent people than an honest, challenging problem whose possible solution will bestow fame and remain as a lasting monument."

Five solutions arrived within a year. Newton solved it overnight, anonymously, but Bernoulli recognized his style: "I recognize the lion by his paw." Leibniz, Jakob Bernoulli (Johann's older

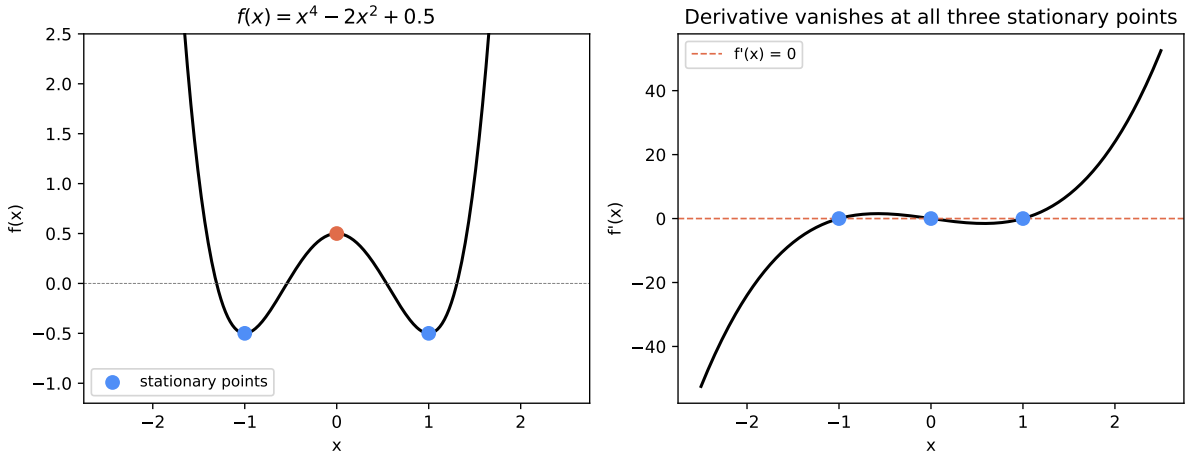


Figure 1: Fermat's stationary point condition: at every local minimum or maximum, the derivative is zero. The derivative is zero at saddle points too — the condition is necessary, not sufficient.

brother), and the Marquis de l'Hôpital also submitted solutions. The answer was the cycloid — the curve traced by a point on the rim of a rolling circle.

The brachistochrone problem was not just a clever puzzle. It was the opening move in the development of the *calculus of variations*, the branch of mathematics that asks: among all possible functions, which one minimizes a given integral? This is optimization over an infinite-dimensional space — not "find the best value of $x$" but "find the best *function* $y(x)$." It is a profound generalization, and it set the agenda for mathematical physics for two centuries.

## Optimization as a Unifying Principle

What makes the history of optimization unusual is how many different fields converged on the same mathematical ideas from completely different directions.

Natural philosophers asked why nature seemed to be economical. Fermat's principle of least time says light takes the path that minimizes travel time. Hamilton's principle says a mechanical system follows the path that minimizes the integral of kinetic minus potential energy — the *action*. Minimum energy, minimum time, minimum cost: the physical world seemed to be optimizing something, and finding what it was minimizing revealed the governing equations.

Economists asked how to allocate limited resources. The question of maximizing utility subject to a budget constraint, or minimizing cost subject to production requirements, turned out to require the same mathematical machinery as finding the minimum of a function subject to inequality constraints.

Engineers asked how to design structures, machines, and systems that were as good as possible within given constraints — minimize weight, maximize strength, minimize propellant mass while meeting performance requirements. The same KKT conditions that describe the optimal allocation of resources in an economy describe the optimal shape of a rocket motor grain.

Military planners, after World War II, asked how to route supplies, schedule operations, and

allocate equipment as efficiently as possible. The result was linear programming — the most widely used optimization technique in the world, handling millions of variables and constraints in every major airline, shipping company, and manufacturing firm on earth.

The common thread is the gradient. Whether you are finding the stationary point of a function, minimizing an integral over curves, satisfying complementary slackness conditions, or following the central path of an interior point method, you are always asking the same question in different notation: in which direction does the objective improve, and how far should we go?

## The Chapters Ahead

The story begins with the brachistochrone and the calculus of variations — the first systematic framework for optimization over function spaces. From there it moves to Lagrange's multiplier, the idea that constrained optimization can be converted to unconstrained optimization by augmenting the objective. Then to Cauchy's steepest descent algorithm of 1847, the first numerical optimization method, rediscovered repeatedly for a century before it became the workhorse of machine learning.

The middle section covers the twentieth-century explosion: Dantzig's simplex method for linear programming, born in the war rooms and operations research centers of 1940s America; Kantorovich's independent discovery of the same ideas in Soviet Russia; and the Kuhn-Tucker conditions for nonlinear programming, which unified constrained optimization theory under a single framework.

The final chapters cover the interior point revolution — Karmarkar's 1984 result that changed the complexity of linear programming and eventually produced the solvers running in every modern engineering tool — and the modern landscape of nonlinear optimization, surrogate methods, and the strange return of gradient descent as the dominant algorithm of an era of machine learning.

The gradient runs through all of it.

---

## Summary

Optimization asks a single question — find the best — and the diversity of contexts in which that question arises has driven the development of mathematics for three centuries. The isoperimetric problem posed by Dido's legend is structurally identical to the grain geometry optimization problem on a modern engineer's workstation. Fermat's observation about stationary points is the ancestor of every gradient-based algorithm. The brachistochrone problem opened the door to the calculus of variations and, through it, to the entire modern theory of optimal control.

The history ahead is a story of ideas generalizing — from functions to functionals, from unconstrained to constrained, from equality constraints to inequality constraints, from smooth to nonsmooth, from small problems solvable by hand to large problems solvable only by computers. Each generalization required new mathematics, and each piece of new mathematics opened new applications.

## Further Reading

Goldstine (1980) is the definitive history of the calculus of variations, tracing the mathematical development from Bernoulli to Weierstrass with full technical detail. Nocedal and Wright (2006) provides modern context: Chapter 1 situates the algorithms of this book within the broader landscape. For the isoperimetric problem specifically, Osserman's *A Survey of Minimal Surfaces* (Dover, 1986) gives a rigorous but readable treatment.

# Preface

Every engineer who has used an optimizer — typed `fmincon` into MATLAB, called `scipy.optimize.minimize`, watched IPOPT grind toward a solution — has encountered the product of roughly three centuries of mathematical labor without knowing it. The algorithm finds the minimum, prints the answer, and moves on. The history behind it stays invisible.

This book is an attempt to make that history visible.

The story of mathematical optimization is not a tidy sequence of breakthroughs. It is a tangle of competing motivations: natural philosophers wondering why light bends the way it does, engineers trying to minimize the weight of a bridge, economists trying to allocate scarce resources, military planners trying to supply an army. The mathematics emerged from all of these pressures at once, and the result — the modern toolkit of gradient descent, linear programming, interior point methods, and constrained nonlinear optimization — is richer for it.

The characters in this story are a strange collection. Joseph-Louis Lagrange, the Italian-born Frenchman who preferred algebra to geometry and gave us the multiplier. Augustin-Louis Cauchy, the most prolific mathematician of the nineteenth century, who invented steepest descent as a throwaway result in 1847 and never followed up on it. George Dantzig, who invented the simplex method on a dare in 1947 and spent decades watching it transform the global economy. Leonid Kantorovich, who independently discovered linear programming in Stalinist Russia, was nearly arrested for the implications, and eventually won the Nobel Prize in Economics. Narendra Karmarkar, a young Indian mathematician at Bell Labs who shocked the operations research world in 1984 with a polynomial-time algorithm for linear programming and became the subject of a legal dispute over intellectual property that would have been unthinkable in any previous mathematical era.

Each of these figures solved a version of the same problem: find the best. Best meaning smallest, or largest, or most efficient, or least costly, depending on who was asking and why. The diversity of applications is part of what makes optimization such a strange and powerful field. The same mathematical idea that tells a soap bubble how to minimize surface area also tells a logistics company how to route ten thousand packages, tells a rocket motor designer how to minimize propellant mass while meeting thrust and pressure constraints, and tells a neural network how to adjust millions of weights to classify images correctly.

The thread connecting all of these is the gradient — the direction of steepest improvement. Follow the gradient down and you find a minimum. Follow it up and you find a maximum. The art of optimization is knowing when following the gradient is enough, and when the terrain is complicated enough that you need something smarter.

This book follows that thread from its origins in seventeenth-century geometry to its present

form in the algorithms running on today's engineering workstations. It is written for engineers and applied mathematicians who use optimization tools and would like to understand where they came from — not as an archaeological curiosity, but because the history illuminates the ideas in ways that textbook derivations often don't.

Each chapter includes Python demonstrations that show the algorithms working on small problems. In the HTML version, these are folded away by default — click "Show Python" to expand them. In the PDF version, they are omitted to keep the reading experience clean. The code is available in Appendix A.

---

*The title comes from a phrase Leonid Kantorovich used in his Nobel lecture: "In search of the optimum, we find not just a number, but a structure." He was talking about linear programming. The observation is general.*

# The Fastest Path

**Learning Objectives**

- Understand the brachistochrone problem and its physical setup
- Derive the Euler-Lagrange equation for functionals
- See how the calculus of variations generalizes derivative-based optimization to function spaces
- Understand why the cycloid is the solution and what it implies about optimal paths

## A Challenge Addressed to the World

Johann Bernoulli published his brachistochrone problem in January 1696 with the theatrical confidence of a man who knew the answer and suspected very few others did. He was right on both counts. The problem — find the curve of fastest descent for a bead sliding without friction between two points — had exactly five correct solutions submitted within the deadline. All five came from the most formidable mathematicians alive.

What made the brachistochrone problem historically significant was not the answer. The cycloid was already known as a curve of elegant properties — Dutch physicist Christiaan Huygens had shown in 1659 that it was also a *tautochrone*, meaning a bead released from any point on a cycloidal arc reaches the bottom in the same time, regardless of starting position. The significance of Bernoulli's challenge was the method required to solve it.

Finding the minimum of a function of a real variable requires calculus. Finding the curve that minimizes a time integral requires something more: a calculus of functions, where the unknown is not a number but a curve. The brachistochrone problem forced mathematicians to develop this calculus, and the result — the Euler-Lagrange equation — became one of the most powerful tools in mathematical physics.

## Setting Up the Problem

A bead starts at the origin with zero velocity and slides without friction along a wire to a point $(x_1, y_1)$, where $y_1 < 0$ (taking downward as positive $y$). Gravity accelerates the bead downward. What shape should the wire take to minimize the total descent time?

The speed of the bead at any height $y$ follows from energy conservation. The kinetic energy gained equals the potential energy lost:

$$\frac{1}{2}mv^2 = mgy \implies v = \sqrt{2gy}$$

A small arc length element along the curve $y(x)$ is $ds = \sqrt{1 + (y')^2}\,dx$. The time to traverse this element is $dt = ds/v$. The total descent time is:

$$T[y] = \int_0^{x_1} \frac{\sqrt{1 + (y')^2}}{\sqrt{2gy}}\,dx$$

This is a *functional* — a function that takes an entire curve $y(x)$ as its input and returns a real number (the descent time). The brachistochrone problem asks: which function $y(x)$ minimizes $T[y]$?

## The Euler-Lagrange Equation

Leonhard Euler, working in the 1740s, developed the systematic method for solving problems of this type. The key insight is to perturb the candidate optimal curve and require that the first-order change in the functional vanish.

Let $y^*(x)$ be the optimal curve and let $\eta(x)$ be any smooth function satisfying $\eta(0) = \eta(x_1) = 0$ (the endpoints are fixed). Consider the perturbed curve $y_\epsilon(x) = y^*(x) + \epsilon\,\eta(x)$. The functional becomes $T(\epsilon) = T[y_\epsilon]$, a function of the scalar $\epsilon$. For $y^*$ to be optimal, we need $dT/d\epsilon = 0$ at $\epsilon = 0$.

For a general functional $J[y] = \int_a^b F(x, y, y')\,dx$, this condition gives the **Euler-Lagrange equation**:

$$\frac{\partial F}{\partial y} - \frac{d}{dx}\frac{\partial F}{\partial y'} = 0$$

This is a necessary condition for an extremum of the functional, exactly analogous to the condition $f'(x) = 0$ for an extremum of an ordinary function. Applied to the brachistochrone functional, where $F = \sqrt{(1 + y'^2)/(2gy)}$, it yields a differential equation whose solution is the cycloid.

## Why the Cycloid Wins

The cycloid's initial steep drop is not a coincidence — it is the optimal trade-off between two competing demands. Falling steeply early converts potential energy to kinetic energy quickly, so the bead is moving fast for most of the journey. But too steep an initial drop means the horizontal component of velocity is sacrificed. The cycloid balances these demands exactly.

A physical intuition: Fermat's principle says light takes the path of least time. When light passes from air into glass, it bends toward the vertical because glass is slower — the light "wants" to spend as little time as possible in the slow medium. The brachistochrone is the same problem in reverse. Gravity plays the role of the refractive index: the bead travels faster at the bottom (where it has fallen further) and slower near the top. The optimal path bends toward the fast region, just as light bends toward the slow region.

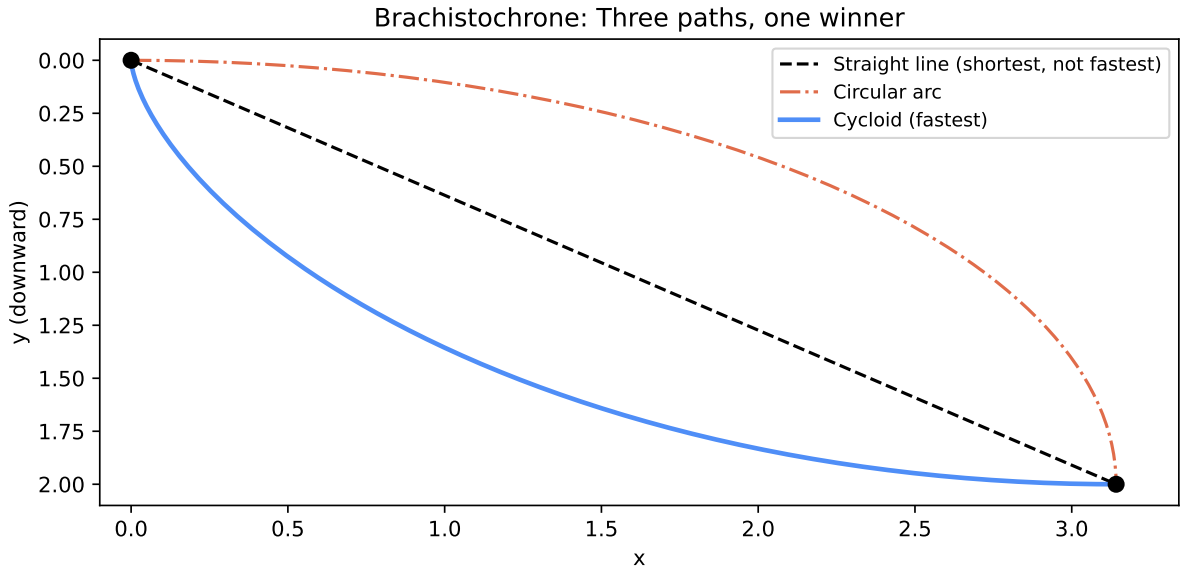


Figure 2: Three paths connecting the same two endpoints. The straight line is shortest but not fastest — gravity is not yet fully engaged. The steep initial drop of the cycloid trades height for speed early, winning the race despite covering more distance.

Johann Bernoulli's own solution used exactly this analogy. He applied Snell's law of refraction to a hypothetical "optical medium" where the refractive index varied with height, and read off the brachistochrone from the path that light would take. It was one of the most elegant solutions in the history of mathematics, and it directly connected optimization to physics in a way that would echo through three centuries of theoretical mechanics.

## Lagrange's Refinement

Joseph-Louis Lagrange, writing in the 1750s as a teenager in Turin, found a more systematic way to derive the Euler-Lagrange equation and extended it to problems with multiple functions and constraints. His approach — now called the method of Lagrange multipliers — will appear in the next chapter. But his contribution to the calculus of variations was equally important: he showed that the Euler-Lagrange equation is not just a trick for specific problems but a general necessary condition for extrema of functionals.

Euler generously renamed the field in Lagrange's honor, calling it the *calculus of variations* (from the Latin for "the calculus of varied curves"), acknowledging that Lagrange's algebraic formulation was cleaner than his own geometric approach. This kind of generosity between mathematicians was characteristic of the eighteenth century and largely disappeared in the nineteenth.

## The Legacy: Hamilton's Principle

The most profound application of the Euler-Lagrange equation was Hamilton's principle of stationary action, published in 1833–1834. William Rowan Hamilton showed that the equations of motion for any mechanical system — Newton's laws in disguise — could be derived as the

Euler-Lagrange equations of a functional, the *action*:

$$S[q] = \int_{t_1}^{t_2} \mathcal{L}(q, \dot{q}, t)\, dt$$

where $\mathcal{L} = T - V$ is the *Lagrangian*, the kinetic energy minus the potential energy. Setting $\delta S = 0$ — requiring the action to be stationary — gives the equations of motion.

Hamilton's principle unified all of Newtonian mechanics under a single variational statement. It later proved to be the foundation of quantum mechanics (Feynman's path integral) and general relativity (the Einstein-Hilbert action). The brachistochrone problem, which seemed like a mathematical game in 1696, had opened a door that led to the deepest structures in physics.

For optimization, the immediate legacy was a framework for handling infinite-dimensional problems — optimization over spaces of functions — that would eventually develop into optimal control theory. The question "what is the optimal trajectory for a spacecraft?" is a brachistochrone problem dressed in rocket fuel.

---

## Summary

The brachistochrone problem forced the development of the calculus of variations — a systematic method for finding functions that minimize or maximize integral functionals. The Euler-Lagrange equation is the variational analogue of setting the derivative equal to zero: a necessary condition for optimality. The cycloid is the solution to the brachistochrone problem, and the intuition behind it — exploit fast regions early, match the geometry to the physics — previews the intuition behind every subsequent optimization algorithm.

Hamilton's reformulation of mechanics as a variational principle revealed that the physical world itself is described by optimization problems. This connection between physics and optimization runs deep enough that Richard Feynman, explaining quantum mechanics to a general audience, used the phrase "nature sniffs out all paths and takes the optimal one." Whether or not that is literally true, it is a useful way to think.

## Further Reading

Goldstine (1980) covers the brachistochrone and the development of the calculus of variations in full technical detail. For Hamilton's principle and its consequences, Goldstein, Poole, and Safko's *Classical Mechanics* (3rd ed., Addison-Wesley, 2002) is the standard reference. Bernoulli's original 1696 paper is reproduced in Struik's *A Source Book in Mathematics, 1200–1800* (Princeton, 1969).

# The Multiplier

**Learning Objectives**

- Understand Lagrange's method of multipliers for constrained optimization
- Derive the first-order necessary conditions for constrained minima
- Interpret the multiplier as the marginal value of relaxing a constraint
- Connect Lagrange's algebraic approach to modern constrained NLP

## A Mathematician Who Preferred Algebra

Joseph-Louis Lagrange was born in Turin in 1736 to an Italian father and a French mother, spent most of his productive life in Berlin and Paris, and managed to remain on good terms with nearly everyone through the French Revolution — no small achievement. He was the most elegant mathematical writer of his century. Where Euler worked by computation, piling equation upon equation until the answer emerged, Lagrange worked by structure, finding the form that made the result obvious.

His *Mécanique Analytique* of 1788 opens with a declaration of aesthetic principle: "One will not find figures in this work. The methods that I expound require neither constructions nor geometrical or mechanical arguments, but only algebraic operations." Lagrange wanted to strip mechanics of its geometric scaffolding and reduce it to pure calculation. He succeeded so completely that modern mechanics is essentially the *Mécanique Analytique* with updated notation.

The multiplier is Lagrange's most famous contribution to optimization, and its elegance is characteristic: a seemingly artificial trick that, once understood, appears not just useful but inevitable.

## The Problem of Constrained Extrema

Find the maximum of a function $f(x, y)$ subject to the constraint $g(x, y) = 0$.

The naive approach — parametrize the constraint, substitute, and differentiate — works when the constraint is simple but becomes unwieldy for complex constraints or multiple variables. Lagrange's approach replaces the constrained problem with an unconstrained one.

Form the *Lagrangian*:

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda\, g(x, y)$$

The necessary conditions for a stationary point are:

$$\frac{\partial \mathcal{L}}{\partial x} = 0, \quad \frac{\partial \mathcal{L}}{\partial y} = 0, \quad \frac{\partial \mathcal{L}}{\partial \lambda} = 0$$

The third condition recovers the constraint $g(x, y) = 0$. The first two give $\nabla f = \lambda \nabla g$: the gradient of the objective must be parallel to the gradient of the constraint at the solution.

The geometric interpretation is clean: at the constrained optimum, the level curves of $f$ are tangent to the constraint surface. If they were not tangent — if they crossed — you could move along the constraint and increase $f$, contradicting optimality.

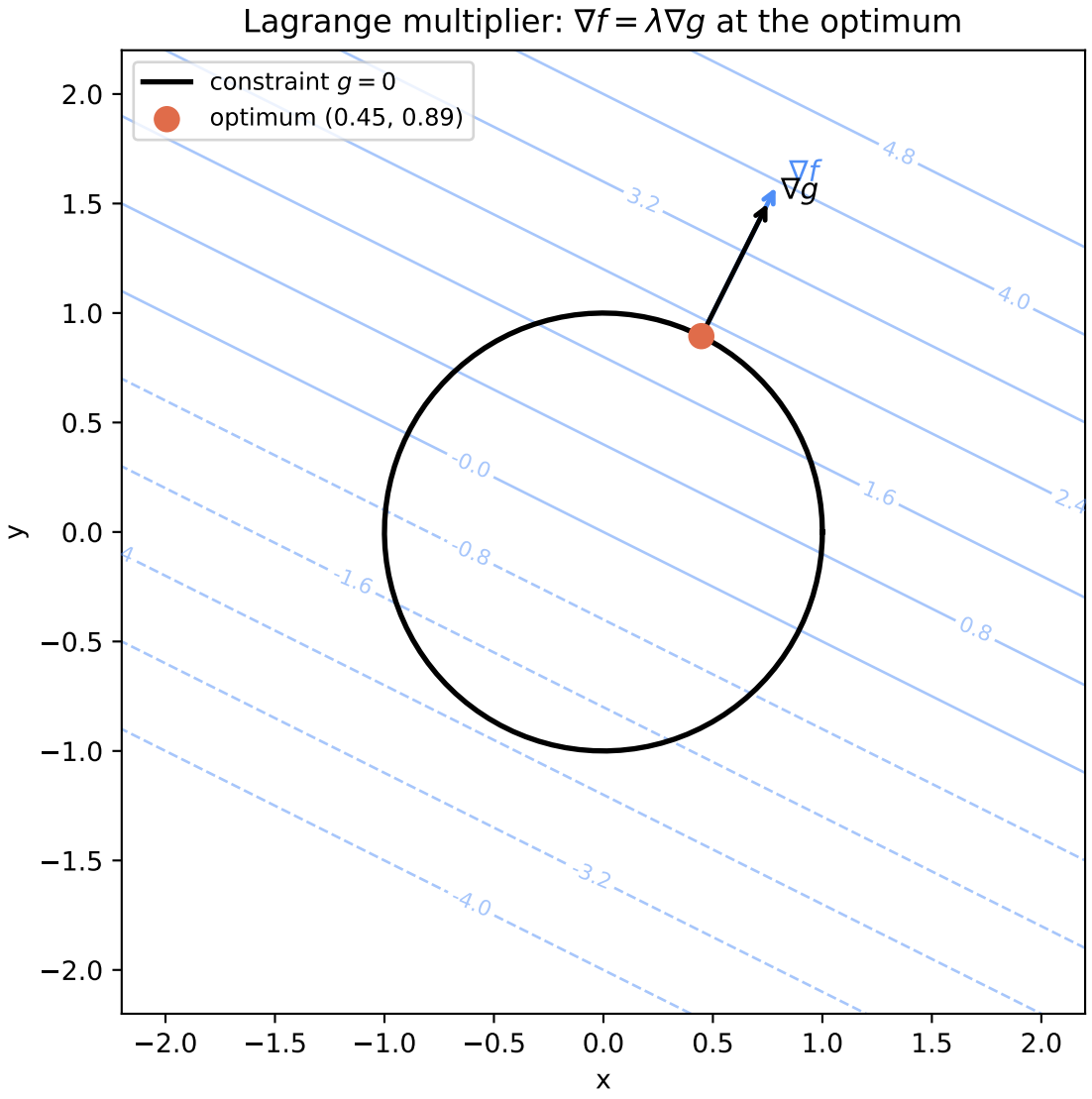


Figure 3: Lagrange multiplier geometry. The optimal point (red dot) is where a level curve of the objective f is tangent to the constraint g = 0. The gradients f and g are parallel there. At any non-optimal feasible point, the gradients are not parallel and movement along the constraint improves the objective.

## The Multiplier as a Price

The Lagrange multiplier $\lambda$ has a natural economic interpretation that Lagrange himself did not emphasize but that was developed thoroughly by economists in the twentieth century.

Consider a manufacturing firm maximizing profit $f(\mathbf{x})$ subject to a resource constraint $g(\mathbf{x}) = b$ (total resource consumption equals available budget). At the optimum, the multiplier $\lambda^*$ equals $\partial f^*/\partial b$ — the rate of change of maximum profit with respect to the resource budget. It is the *shadow price* of the constraint: how much additional profit the firm could earn if the budget were relaxed by one unit.

This interpretation makes the multiplier concrete. A multiplier near zero means the constraint is not actively limiting — relaxing it would barely help. A large positive multiplier means the constraint is binding tightly — a small relaxation would yield a large gain. In engineering, multipliers on stress constraints tell you which structural member is limiting the design; multipliers on pressure constraints tell you how much performance you are leaving on the table by staying within the operating limit.

## Extending to Inequality Constraints

Lagrange's original formulation handles equality constraints. Engineering problems almost always involve inequality constraints — the chamber pressure must not *exceed* 2,500 psi, not equal it. Extending the multiplier method to inequalities was the work of the twentieth century, and it produced the Kuhn-Tucker conditions discussed in Chapter 7.

The key complication: an inequality constraint $g(\mathbf{x}) \leq 0$ may or may not be active at the optimum. If it is inactive ($g(\mathbf{x}^*) < 0$), it does not constrain the solution and the corresponding multiplier is zero. If it is active ($g(\mathbf{x}^*) = 0$), it acts like an equality constraint with a nonnegative multiplier.

This complementary behavior — either the constraint is active or the multiplier is zero, but not both — is complementary slackness, and it is the heart of modern constrained optimization theory.

## Lagrange in Modern Solvers

Every modern constrained optimization solver — SQP, interior point, augmented Lagrangian — maintains estimates of the Lagrange multipliers alongside the design variables. The multipliers are dual variables; the design variables are primal. Updating both together is what makes these methods efficient.

The augmented Lagrangian method, developed in the 1960s by Hestenes and Powell, adds a penalty term to the Lagrangian to improve conditioning:

$$\mathcal{L}_\rho = f(\mathbf{x}) - \lambda g(\mathbf{x}) + \frac{\rho}{2} g(\mathbf{x})^2$$

The penalty parameter $\rho$ ensures that the augmented Lagrangian is strongly convex near the solution, making it easy to minimize by gradient methods. Iterating between minimizing $\mathcal{L}_\rho$ over $\mathbf{x}$ and updating $\lambda$ produces the alternating direction method of multipliers (ADMM), now one of the dominant algorithms in distributed optimization and machine learning.

## Summary

Lagrange's multiplier converts a constrained optimization problem into an unconstrained one by augmenting the objective with a weighted constraint term. The weight — the multiplier — equals the shadow price of the constraint at the optimum: how much the objective would improve if the constraint were relaxed. The geometric condition is tangency: level curves of the objective are tangent to the constraint surface at the solution. Every modern constrained optimization method maintains multiplier estimates and uses them to guide the search.

## Further Reading

Lagrange (1788) is the original source. For a modern treatment of multipliers and their economic interpretation, Luenberger's *Linear and Nonlinear Programming* (4th ed., Springer, 2016) is authoritative. Boyd and Vandenberghe's *Convex Optimization* (Cambridge, 2004) covers the augmented Lagrangian and ADMM in depth; the full text is freely available online.

# Going Downhill

**Learning Objectives**

- Understand Cauchy's steepest descent algorithm and its 1847 origins
- Analyze convergence behavior and the zigzag phenomenon
- Understand why steepest descent was rediscovered repeatedly for a century
- Connect the historical algorithm to modern gradient descent in machine learning

## Cauchy's Throwaway Result

In 1847, Augustin-Louis Cauchy published a short paper in the *Comptes Rendus de l'Académie des Sciences* with the title "Méthode générale pour la résolution des systèmes d'équations simultanées" — General method for the resolution of systems of simultaneous equations. The paper presented what we now call the steepest descent algorithm as a side result, mentioned almost in passing. Cauchy did not follow up on it. He did not analyze its convergence. He did not apply it to any interesting problem.

He had, however, invented the method that would become the most important algorithm in the history of machine learning, used today to train every large neural network on earth.

The algorithm is simple: to minimize a function $f(\mathbf{x})$, start at some point $\mathbf{x}_0$ and repeatedly move in the direction of the negative gradient:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$$

The gradient $\nabla f$ points uphill; its negative points downhill. The step size $\alpha_k$ (the learning rate, in modern parlance) controls how far to go. The idea is obvious in retrospect — go downhill — and its obvious simplicity obscured the nontrivial question of how fast it converges and when it fails.

## The Zigzag Problem

For a smooth, strongly convex function, steepest descent converges to the minimum. But the convergence can be painfully slow, and the reason is geometric.

Consider minimizing a quadratic $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$ where $A$ is symmetric positive definite. The level curves are ellipses. Steepest descent with exact line search takes steps that are exactly orthogonal at each iteration — the new gradient is perpendicular to the previous step. On a

round ellipse (eigenvalues of $A$ close together), this works well. On a very elongated ellipse (eigenvalues far apart), the algorithm zigzags back and forth across the valley, making tiny net progress with each step.

The convergence rate depends on the *condition number* $\kappa = \lambda_{max}/\lambda_{min}$ — the ratio of largest to smallest eigenvalue of $A$. The error after $k$ iterations satisfies:

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^k \|\mathbf{x}_0 - \mathbf{x}^*\|$$

For $\kappa = 10$, the factor is $(9/11)^k \approx 0.82^k$ — slow. For $\kappa = 1000$, it is $(999/1001)^k \approx 0.998^k$ — catastrophically slow. This is the curse of ill-conditioning, and it is why steepest descent alone is almost never used for serious numerical optimization.

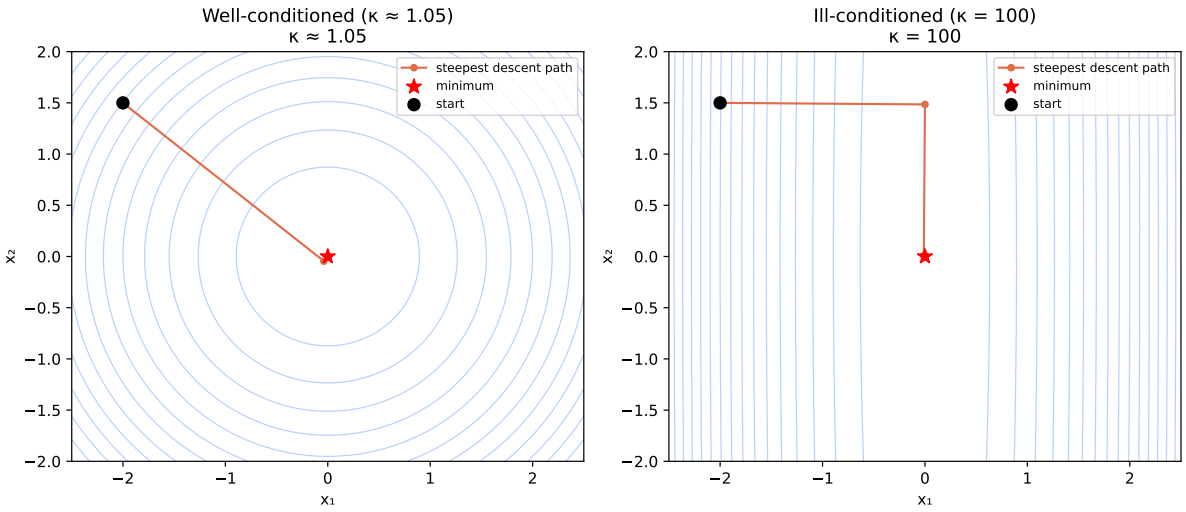


Figure 4: Steepest descent on a well-conditioned (left) and ill-conditioned (right) quadratic. On the elongated ellipse, the algorithm zigzags, making tiny net progress toward the minimum (red star). The condition number governs the severity of the zigzag.

## A Century of Rediscovery

Cauchy's 1847 paper was largely forgotten. The method was rediscovered — independently and repeatedly — by mathematicians who did not know the prior work. The most important rediscovery was by Haskell Curry in 1944, working in the context of electronic computing. Curry wrote a technical report for the Ballistic Research Laboratory analyzing the convergence of gradient methods for solving systems of equations, unaware of Cauchy's paper.

The pattern of independent rediscovery is itself historically interesting. The steepest descent idea is simple enough to be reinvented from scratch by anyone who has thought carefully about the geometry of optimization, but it was not published in places where applied mathematicians would encounter it. The *Comptes Rendus* was not widely read outside France; Cauchy's paper was buried in his vast output; and the idea of numerical iterative methods was not yet a recognized field with its own literature.

The situation changed in the 1950s. The development of digital computers created a pressing demand for practical algorithms for solving large systems of equations and optimization problems. Gradient methods were rediscovered, analyzed, and published in journals that numerical analysts read. By 1960, the literature on gradient methods was growing rapidly, and the question of convergence rates — how fast the method approaches the minimum — was being attacked with serious mathematical tools.

## Conjugate Gradient: The Fix That Worked

The zigzag problem was solved in 1952 by Magnus Hestenes and Eduard Stiefel, working independently at the National Bureau of Standards and ETH Zürich. The conjugate gradient method chooses search directions that are mutually *conjugate* with respect to the Hessian matrix, meaning $\mathbf{d}_i^T A \mathbf{d}_j = 0$ for $i \neq j$. This property ensures that minimizing along each direction does not undo progress made along previous directions.

The result: for a quadratic function in $\mathbb{R}^n$, conjugate gradient finds the exact solution in at most $n$ steps. The zigzag is eliminated. For non-quadratic functions, conjugate gradient is used as an iterative method with restarts, and it remains one of the most efficient large-scale optimization algorithms known.

## The Machine Learning Connection

In 2012, Geoffrey Hinton and colleagues published the AlexNet neural network, which won the ImageNet image classification competition by a margin that stunned the computer vision community. AlexNet was trained using stochastic gradient descent — a variant of Cauchy's 1847 algorithm where the gradient is estimated from a random mini-batch of training data rather than the full dataset.

The irony is that the algorithm used to train the most consequential AI systems of the early twenty-first century was invented 165 years earlier as a footnote in a paper about solving linear systems. Cauchy did not think it was interesting. He was wrong.

The reason stochastic gradient descent works for neural network training is that neural networks are so overparameterized — they have far more parameters than data points — that noise in the gradient estimate is not a bug but a feature. The noise prevents the optimizer from settling into sharp, poorly-generalizing minima and encourages it toward flat, broad minima that generalize better. This is an empirical observation that is not yet fully understood theoretically, and it would have baffled Cauchy entirely.

---

## Summary

Steepest descent — moving in the direction of the negative gradient — is the oldest numerical optimization algorithm, invented by Cauchy in 1847 and rediscovered repeatedly for a century. It converges for smooth strongly convex functions but suffers from the zigzag phenomenon on ill-conditioned problems, with convergence rate governed by the condition number of the Hessian. The conjugate gradient method of Hestenes and Stiefel (1952) fixed the zigzag by choosing mutually conjugate search directions, finding the exact minimum of a quadratic in at most $n$

steps. Modern machine learning runs on stochastic gradient descent — Cauchy's algorithm applied to functions with millions of variables and billions of data points.

## Further Reading

Cauchy (1847) is the original 1847 paper, two pages long. Hestenes and Stiefel's 1952 paper "Methods of Conjugate Gradients for Solving Linear Systems" (*Journal of Research of the National Bureau of Standards*, 49:409–436) introduced conjugate gradient. For the modern machine learning context, LeCun, Bottou, Orr, and Müller's chapter "Efficient BackProp" in *Neural Networks: Tricks of the Trade* (Springer, 1998, 2012) connects Cauchy's algorithm to deep learning practice.

# The War Problem

> **Learning Objectives**
>
> - Understand the wartime origins of linear programming
> - Grasp the simplex method geometrically and algorithmically
> - Understand why LP became the dominant optimization tool of the twentieth century
> - Appreciate Dantzig's role and the institutional context of his discovery

## The Problem of Supplying an Army

In June 1947, a young statistician named George Dantzig walked into the Pentagon with a problem. He had been working for the U.S. Air Force on the question of how to mechanize the planning process — how to take the complex logistical problem of deploying, supplying, and scheduling military forces and turn it into something that could be solved systematically rather than by guesswork and experience.

The core mathematical problem was: minimize a linear function of many variables subject to a system of linear inequality constraints. Dantzig called it linear programming. He did not know at the time that a Soviet economist named Leonid Kantorovich had independently formulated the same problem eight years earlier, or that the mathematical structure had been studied by nineteenth-century French mathematicians in a completely different context. He also did not know that within a decade, the technique he was developing would be used to route every major airline's fleet, schedule every major refinery's production, and plan the supply chains of companies that had not yet been founded.

Dantzig invented the simplex method that summer, working alone. He presented it to the economist John von Neumann in October 1947. Von Neumann listened for a minute, then began pacing and lecturing about duality theory — the mathematical relationship between a linear program and its mirror image — which he had apparently worked out on the spot in the course of the meeting. "I began to feel uneasy," Dantzig wrote later, "in case Johnny was going to work out too much of it before I had a chance to do it myself."

## Linear Programs and Their Geometry

A linear program in standard form minimizes $\mathbf{c}^T\mathbf{x}$ subject to $A\mathbf{x} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$, where $\mathbf{c} \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$.

The feasible set is a convex polytope — a many-sided geometric solid. The objective $\mathbf{c}^T\mathbf{x}$ is

linear, so its level sets are parallel hyperplanes. The minimum occurs at a vertex of the polytope — a corner point where $n$ constraints are active simultaneously. (It also occurs at a vertex for the maximum, or along an edge if two adjacent vertices achieve the same optimal value.)

This observation is the geometric basis of the simplex method: since the optimum is at a vertex, search only the vertices. Move from vertex to adjacent vertex, improving the objective at each step, until no adjacent vertex is better. Stop.

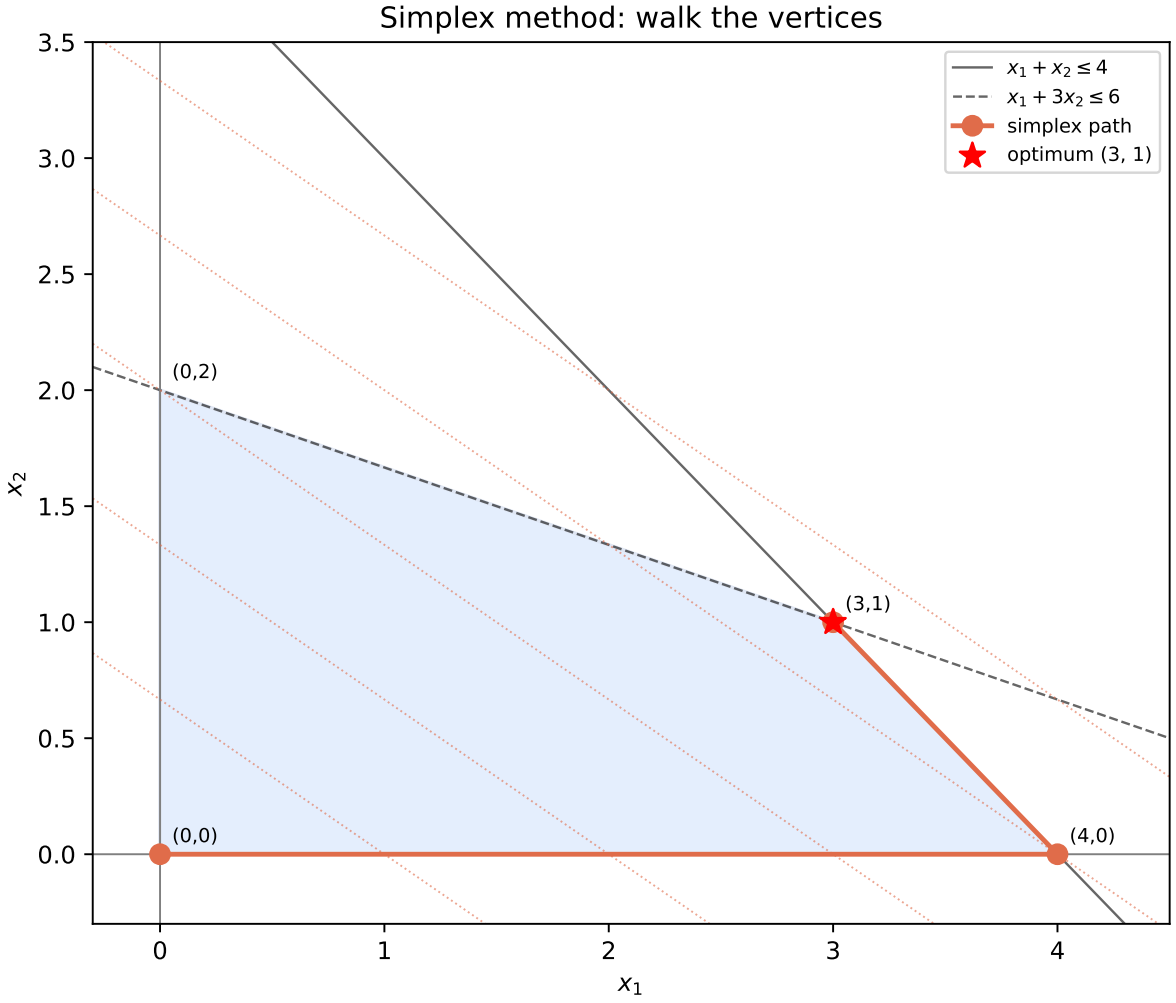


Figure 5: The simplex method in 2D. The feasible polytope (shaded) is bounded by the constraint hyperplanes. The objective contours (dashed parallel lines) move until they are tangent to the polytope. The optimum is at a vertex. The simplex path walks along edges from vertex to vertex, always improving the objective.

## The Simplex Algorithm

The simplex algorithm is, in its basic form, elegant and simple. Each step:

1. **Current vertex:** maintain a *basic feasible solution* — a vertex of the polytope, identified by a set of $m$ active constraints (basic variables).

2. **Reduced costs:** compute the rate of change of the objective for each non-basic variable. A negative reduced cost (for minimization: positive for a non-basic variable that could increase) indicates that moving in that direction improves the objective.
3. **Pivot:** if any reduced cost indicates improvement, choose one such variable to enter the basis (become active) and another to leave (become inactive). Move to the adjacent vertex.
4. **Optimality:** if no reduced cost indicates improvement, the current vertex is optimal.

The pivot operation is a matrix operation on the basis matrix. In Dantzig's formulation, it is equivalent to a Gaussian elimination step on the constraint matrix. The full algorithm is a sequence of such steps.

## An Exponential Algorithm That Runs in Polynomial Time

The simplex method has an uncomfortable theoretical property: in the worst case, it visits every vertex of the polytope before finding the optimum. The number of vertices can be exponential in the problem dimension — Klee and Minty constructed an explicit example in 1972 where the simplex method visits all $2^n$ vertices of an $n$-dimensional polytope. In theory, the simplex method is exponentially slow.

In practice, it almost never is. The reason is not fully understood. Empirical evidence suggests that the simplex method visits $O(m)$ vertices on typical problems — a number proportional to the number of constraints, not the dimension. Various theoretical explanations have been proposed, involving smoothed analysis (the method is polynomial on randomly perturbed instances) and probabilistic arguments, but none fully explains the observed behavior.

The gap between worst-case and average-case performance is one of the stranger features of the simplex method. It is exponential by construction and polynomial in practice, and the mathematical community spent thirty years after Klee and Minty trying to understand why.

## LP in the World

By 1960, linear programming was being used to plan oil refinery operations, optimize airline crew scheduling, route military supply chains, and manage agricultural production. The problems being solved were large by the standards of the day — thousands of variables and constraints — and required mainframe computers to solve in reasonable time.

The economic impact was enormous. A refinery that runs a daily LP to optimize the allocation of crude oil to products might save several percent of its operating cost. At the scale of a major refinery, that is tens of millions of dollars per year. Airlines solved LP problems to minimize fuel cost, crew hours, and aircraft routing simultaneously. The U.S. military used LP to plan the logistical support of every major operation from Korea through Vietnam.

Dantzig received the National Medal of Science in 1975, the John von Neumann Theory Prize in 1975, and the Harold Pender Award. He did not receive the Nobel Prize. His collaborator Leonid Kantorovich did, in 1975 — the same year.

---

## Summary

Linear programming formulates optimization as minimizing a linear objective over a polytope defined by linear inequalities. The optimum is at a vertex. The simplex method walks from vertex to adjacent vertex, improving the objective at each step, and terminates when no adjacent vertex is better. It is worst-case exponential but average-case polynomial, and it became the dominant optimization tool of the twentieth century within a decade of its invention. George Dantzig invented it in 1947, building on wartime operations research and the emerging theory of linear inequalities.

## Further Reading

Dantzig (1963) is the primary source: Dantzig's own account of the method and its history. Schrijver (1998) provides the mathematical context including Kantorovich's independent work. For the human story, Dantzig's memoir "Linear Programming" in *Operations Research*, 50(1):42–47 (2002), is short and readable.

# The Soviet Optimist

**Learning Objectives**

- Understand Kantorovich's independent discovery of linear programming in 1939
- Appreciate the political context that suppressed and delayed his work
- Understand the transportation problem and its historical significance
- Connect the Soviet planning context to the modern interpretation of LP duality

## A Problem from the Plywood Trust

In 1938, Leonid Kantorovich was a twenty-six-year-old professor at Leningrad University, already recognized as one of the leading mathematicians in the Soviet Union. He was asked by the Laboratory of the Plywood Trust — a Soviet state enterprise — to help with a practical problem: how to distribute the workload across their machines to maximize output, given that different machines were better suited for different tasks and different raw materials.

The problem was, in modern terms, a linear program. Kantorovich recognized this immediately. More importantly, he recognized that it was one instance of a general class of problems — the class we now call linear programming — and that the same mathematical structure appeared in many different economic and industrial contexts. He wrote up his results in a 1939 monograph titled *Mathematical Methods of Organizing and Planning Production*, which was published in a small edition by Leningrad University Press and promptly disappeared.

The monograph disappeared for a reason. Its subject was dangerous.

## The Political Calculus

Soviet economic planning in the 1930s was not, in principle, opposed to mathematics. Stalin's industrialization drive required technically trained personnel, and the Soviet educational system invested heavily in science and engineering. Mathematics was respected, even celebrated.

But economic theory was another matter. The mathematical theory of optimal allocation of resources — the idea that prices, or shadow prices, or any quantitative signals could guide economic decisions — was ideologically fraught. Marxist economics held that prices under capitalism were instruments of exploitation, not coordination mechanisms. The idea that a planning economy needed price-like signals to function efficiently was, to doctrinaire Marxists, a concession to bourgeois economics that bordered on treason.

Kantorovich's results implied, inescapably, that optimal resource allocation produced shadow prices — the Lagrange multipliers of the planning problem. These multipliers had the same mathematical structure as market prices. Publishing this observation prominently in Stalin's Soviet Union was not obviously wise.

Kantorovich navigated this carefully. His 1939 monograph framed the results in terms of "resolving multipliers" rather than prices, and emphasized practical computation rather than economic interpretation. He knew what the multipliers meant. He was careful about saying so.

## The Transportation Problem

The most elegant problem in Kantorovich's 1939 monograph was the transportation problem. Suppose there are $m$ sources, each with a known supply of some commodity, and $n$ destinations, each with a known demand. The cost of shipping one unit from source $i$ to destination $j$ is $c_{ij}$. Find the shipping plan that minimizes total cost while satisfying all supply and demand constraints.

This is a linear program with $mn$ variables (the amounts shipped on each route) and $m + n$ constraints (supply at each source, demand at each destination). Its special structure — the constraint matrix has a particularly simple form, with each variable appearing in exactly two constraints — makes it far easier to solve than a general LP.

The transportation problem appears throughout logistics, supply chain management, and economic geography. It is also the mathematical backbone of the *optimal transport* theory developed by Gaspard Monge in 1781 and Kantorovich in 1942, which has become one of the most active research areas in modern mathematics and machine learning (the Wasserstein distance between probability distributions is an optimal transport problem).

## Rediscovery, Suppression, and the Nobel

Kantorovich continued working on linear programming through the 1940s and 1950s, developing what he called the method of *resolving multipliers* — equivalent to the simplex method, though he arrived at it by different reasoning. His 1942 paper on optimal transport was published during the siege of Leningrad. He had, by that point, been working on optimization methods for more than a decade, with almost no contact with Western mathematics.

When Dantzig's work became known in the Soviet Union in the early 1950s, Soviet mathematicians quickly recognized that Kantorovich had priority on the fundamental ideas. This was a politically sensitive situation: acknowledging a Soviet mathematician's independent discovery was a point of national pride, but engaging seriously with Western academic work required institutional permissions that were difficult to obtain.

Kantorovich eventually published a comprehensive treatment of linear programming in Russian in 1959. The English translation appeared in 1960. Western economists, encountering his work for the first time, were struck by two things: its mathematical quality, and the care with which he had avoided discussing the ideological implications of shadow prices.

In 1975, Kantorovich shared the Nobel Prize in Economics with Tjalling Koopmans (who had independently developed linear programming in the Western context) for "contributions to

the theory of optimum allocation of resources." The Nobel committee's phrasing was careful. Kantorovich had spent thirty-five years being careful too.

---

## Summary

Leonid Kantorovich independently discovered linear programming in 1939 — eight years before Dantzig — while working on a practical industrial problem in Leningrad. The political context of Stalinist Soviet Union suppressed the publication and dissemination of his work; its mathematical content implied the existence of price-like signals in planned economies, a politically dangerous observation. Kantorovich developed the transportation problem and the method of resolving multipliers, equivalent to the simplex method, and eventually received the Nobel Prize in Economics in 1975. His story is a reminder that mathematical ideas do not develop in political vacuums.

## Further Reading

Kantorovich (1960) is the English translation of his major work. For the political and intellectual context, Aron Katsenelinboigen's *Studies in Soviet Economic Planning* (Sharpe, 1978) is informative. Cédric Villani's *Optimal Transport: Old and New* (Springer, 2009) traces Kantorovich's optimal transport work to its modern consequences in mathematics and machine learning.

# The Conditions

**Learning Objectives**

- Derive the Karush-Kuhn-Tucker conditions for constrained nonlinear optimization
- Understand primal feasibility, dual feasibility, and complementary slackness
- Appreciate Karush's overlooked 1939 master's thesis
- Connect KKT theory to modern NLP solvers

## Three Names, One Theorem

The Karush-Kuhn-Tucker conditions — the fundamental optimality conditions for constrained nonlinear programming — have a complicated naming history that reflects the sociology of mid-twentieth-century mathematics.

William Karush derived the conditions in his master's thesis at the University of Chicago in 1939. The thesis was not published in a journal. It was not widely circulated. Karush went on to a career in applied mathematics at RAND and System Development Corporation, and the thesis remained obscure.

Harold Kuhn and Albert Tucker, working at Princeton, independently derived the same conditions in 1950 and presented them at the Second Berkeley Symposium on Mathematical Statistics and Probability in 1951. Their paper was published in the conference proceedings. It became one of the most influential papers in the history of applied mathematics.

For decades, the conditions were called the Kuhn-Tucker conditions. The historical correction — acknowledging Karush's priority — was slow to propagate. The triple name "KKT" became standard in the 1990s, fifty years after Karush's original derivation. Karush died in 1997, having lived long enough to see his contribution acknowledged, if not quite in the form he might have hoped.

## The KKT Conditions

Consider the nonlinear program:

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad g_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, m; \quad h_j(\mathbf{x}) = 0,\ j = 1, \ldots, p$$

Under regularity conditions (constraint qualification), necessary conditions for a local minimum $\mathbf{x}^*$ are the existence of multipliers $\mu \geq \mathbf{0}$ and $\lambda$ such that:

**Stationarity:**

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{m} \mu_i \nabla g_i(\mathbf{x}^*) + \sum_{j=1}^{p} \lambda_j \nabla h_j(\mathbf{x}^*) = \mathbf{0}$$

**Primal feasibility:**

$$g_i(\mathbf{x}^*) \leq 0, \quad h_j(\mathbf{x}^*) = 0$$

**Dual feasibility:**

$$\mu_i \geq 0 \quad \forall i$$

**Complementary slackness:**

$$\mu_i g_i(\mathbf{x}^*) = 0 \quad \forall i$$

The last condition is the key one. Either the constraint is active ($g_i = 0$) and the multiplier can be positive, or the constraint is inactive ($g_i < 0$) and the multiplier must be zero. The constraint and its multiplier cannot both be positive.

```
Optimal x: (0.5000, 0.5000)
f(x*) = 0.2500
g1(x*) = 0.0000  (active if ~0)
g2(x*) = 0.3000  (active if ~0)
Constraint 1 active: True
Constraint 2 active: False
```

Figure 6

## What the Conditions Actually Say

The stationarity condition says the gradient of the objective, at the optimum, must be a nonnegative linear combination of the active constraint gradients. Geometrically: the objective cannot point into the feasible set at the optimum — if it did, you could improve the objective by moving slightly into the feasible set, contradicting optimality.

Complementary slackness says an inactive constraint does not constrain the solution. Its multiplier is zero because relaxing it — pushing the boundary slightly further away — would not change the optimum. An active constraint's multiplier measures how much the objective would improve if the constraint boundary were relaxed.

Together, these conditions characterize the constrained optimum just as $f'(x) = 0$ characterizes the unconstrained optimum. They are necessary conditions under regularity assumptions (constraint qualifications ensure the multipliers exist). They are also sufficient conditions when the objective is convex and the feasible set is convex.

## Constraint Qualifications

The KKT conditions hold under regularity conditions called constraint qualifications. The most common is linear independence constraint qualification (LICQ): the gradients of the active constraints at the solution are linearly independent. When this fails — when active constraints are redundant or degenerate — multipliers may not be unique or may not exist.

Constraint qualification failures are a common source of solver trouble in practice. A pressure constraint and a temperature constraint that are always simultaneously active because they share the same physical mechanism may produce a numerically rank-deficient constraint Jacobian, causing interior point solvers to report "restoration phase failure" and SQP solvers to report abnormal terminations. The fix is usually to identify and remove the redundant constraint.

## The Road to Sufficiency: Convexity

The KKT conditions are necessary for any smooth NLP under regularity. For convex programs — convex objective and convex feasible set — they are also sufficient. Any KKT point is a global minimum.

This sufficiency result makes convex programming a qualitatively different field from general nonlinear programming. Gradient-based methods converge to global optima, not just local ones. The duality gap — the difference between the primal optimum and the dual optimum — is zero. The multipliers have a clean economic interpretation. Many structural problems in engineering (least-norm solutions, portfolio optimization, certain geometric design problems) are convex programs, and recognizing this structure is a significant practical advantage.

---

## Summary

The Karush-Kuhn-Tucker conditions provide necessary (and for convex problems, sufficient) conditions for constrained optima. Stationarity requires the gradient of the objective to be a nonnegative combination of active constraint gradients. Dual feasibility requires multipliers on inequality constraints to be nonnegative. Complementary slackness requires that inactive constraints have zero multipliers. The conditions were derived by Karush in 1939 (unpublished) and Kuhn and Tucker in 1951 (published), and are the foundation of every constrained optimization algorithm in use today.

## Further Reading

Kuhn and Tucker (1951) is the 1951 paper. Karush's original 1939 thesis is reproduced in Kjeldsen's "A Contextualized Historical Analysis of the Kuhn-Tucker Theorem in Nonlinear Programming" (*Historia Mathematica*, 27(4):331–361, 2000), which also documents the history of the naming controversy. Nocedal and Wright (2006) Chapter 12 provides a thorough modern treatment of KKT theory and constraint qualifications.

# The Interior Revolution

**Learning Objectives**

- Understand Karmarkar's 1984 polynomial-time LP algorithm and why it was revolutionary
- Grasp the complexity-theoretic significance of polynomial vs. exponential algorithms
- Understand the AT&T context and the patent controversy
- Connect Karmarkar's algorithm to the modern interior point methods in IPOPT

## The Announcement

On August 13, 1984, Narendra Karmarkar submitted a paper to the journal *Combinatorica* titled "A new polynomial-time algorithm for linear programming." He was 28 years old, working at Bell Laboratories in Murray Hill, New Jersey. The paper was seven pages long.

Bell Labs held a press conference the following month. This was unusual. Bell Labs did not typically hold press conferences for pure mathematics papers. But Karmarkar's result, combined with Bell's aggressive publicity, produced one of the most remarkable media events in the history of applied mathematics. The *New York Times* ran a front-page story. *Business Week* ran a cover. The claim was that Karmarkar's algorithm would solve linear programs fifty to one hundred times faster than the simplex method, and that this would transform every industry that used LP.

The reaction from the operations research community was a mixture of excitement and skepticism. The excitement was genuine: a polynomial-time algorithm for LP had theoretical implications that went well beyond practical speed. The skepticism was also genuine: the simplex method, despite its exponential worst case, had decades of practical refinement behind it, and raw algorithmic speed in benchmark tests was not the same as practical superiority on real problems.

## Polynomial Time and What It Means

The complexity theory context is necessary to understand why Karmarkar's result was theoretically significant regardless of practical speed.

An algorithm runs in polynomial time if its running time is bounded by a polynomial in the size of the input. An algorithm runs in exponential time if its running time can grow exponentially with input size. For small inputs, exponential algorithms can be fast. For large inputs, they

become intractable — there is no hardware improvement that compensates for exponential growth.

The simplex method, as Klee and Minty had shown in 1972, has exponential worst-case complexity. It visits $2^n$ vertices in the worst case. For $n = 100$ design variables, this is $10^{30}$ iterations — more than the age of the universe at a trillion iterations per second. In practice, Klee-Minty instances never appear, and the simplex method solves real problems efficiently. But the theoretical guarantee of polynomial time had been missing.

Karmarkar provided it. His algorithm solves an LP in $O(n^{3.5}L)$ arithmetic operations, where $n$ is the number of variables and $L$ is the number of bits needed to encode the input. The polynomial guarantee means that as problems scale, the algorithm remains tractable — a property the simplex method cannot claim in theory.

## The Algorithm

Karmarkar's algorithm belongs to the family of interior point methods. Rather than walking along the boundary of the feasible polytope (as simplex does), it moves through the interior, approaching the optimum along the central path.

The key idea: transform the LP so that the current iterate is at the center of the feasible set, apply a projective transformation that maps the feasible set to a standard simplex, take a step along the projected gradient, and invert the transformation. At each step, the algorithm makes geometric progress — reduces the objective by a constant fraction of the gap to optimality.

Karmarkar's specific transformation was projective (using the full projective group), which was more general than necessary. Later researchers showed that affine scaling — a simpler transformation — achieved the same polynomial complexity and was easier to implement. The modern interior point methods in practical solvers (IPOPT, KNITRO, Mosek) descend from these primal-dual algorithms, which were developed in the late 1980s and early 1990s and are significantly more efficient than Karmarkar's original formulation.

## The Patent and the Controversy

In January 1988, AT&T filed a patent on Karmarkar's algorithm. This was not, by the standards of the time, unusual — Bell Labs patented algorithms routinely. But it created a storm of controversy in the mathematical community.

The operations research community depended on linear programming solvers. Several commercial solvers implemented Karmarkar-type algorithms. If AT&T enforced the patent aggressively, it could restrict academic and commercial use of interior point methods. The debate about patenting mathematical algorithms — which had been ongoing since the 1970s — became more pointed.

In practice, AT&T did not enforce the patent broadly, and it expired in 2006. But the controversy shaped attitudes in the optimization community toward intellectual property and open-source software. The development of IPOPT as a fully open-source implementation, released by Wächter and Laird in 2006, was in part a response to the perceived risk of patent encumbrances on commercial solvers.

## From LP to NLP: The Legacy

Karmarkar's algorithm, and the interior point revolution it triggered, turned out to be far more consequential for nonlinear programming than for linear programming. The simplex method remained competitive for LP in practice (CPLEX's simplex implementation is faster than its interior point implementation on many LP instances). But for NLP — where the feasible set is curved and the simplex method does not apply — interior point methods are dominant.

The extension from LP to NLP, accomplished by Mehrotra, El-Ghaoui, Vanderbei, and others in the early 1990s, and culminating in Wächter and Laird's IPOPT in 2006, gave engineers a solver that could handle problems with hundreds of thousands of variables and sparse constraint Jacobians — problems that SQP methods could not approach. Structural optimization, power grid optimization, trajectory optimization, and computational chemistry all became tractable.

The lesson is characteristic of the history of optimization: a theoretical advance in a simple setting (LP complexity) generates practical tools in a much harder setting (NLP for engineering) through a decade of subsequent engineering work.

---

## Summary

Karmarkar's 1984 paper established the first polynomial-time algorithm for linear programming, launching the interior point revolution. The algorithm moves through the interior of the feasible polytope rather than along its boundary, achieving geometric progress toward the optimum at each step. The AT&T patent created controversy but did not prevent the method's development. The true legacy of Karmarkar's work was not in LP — where simplex remains competitive — but in NLP, where interior point methods now dominate large-scale constrained optimization.

## Further Reading

Karmarkar (1984) is the original paper. For the controversy and context, Robert Bland's "New Finite Pivoting Rules for the Simplex Method" (*Mathematics of Operations Research*, 1977) and the discussions in Schrijver (1998) provide background. For the path from Karmarkar's LP algorithm to modern NLP interior point methods, Nocedal and Wright (2006) Chapter 19 is the standard reference.

# The Modern Toolbox

**Learning Objectives**

- Understand how SQP and trust-region methods handle general NLP
- Understand IPOPT's role in large-scale engineering optimization
- See how algorithmic ideas from different eras combine in modern solvers
- Connect the historical development to the solvers available today

## From Algorithms to Solvers

By 1990, the theoretical foundations of optimization were largely in place. The calculus of variations (Euler, Lagrange, Hamilton) handled infinite-dimensional problems. Linear programming (Dantzig, Kantorovich) handled large linear problems efficiently. The KKT conditions (Karush, Kuhn, Tucker) characterized constrained optima for general smooth NLPs. Interior point methods (Karmarkar and successors) provided polynomial algorithms for LP and, increasingly, NLP. Quasi-Newton methods (Broyden, Fletcher, Goldfarb, Shanno — the BFGS formula of 1970) handled medium-scale unconstrained problems.

What remained was engineering: turning these theoretical algorithms into robust, reliable software that could handle the full range of messy, ill-conditioned, poorly-scaled problems that engineers actually needed to solve.

The 1990s and 2000s were the era of solvers. SNOPT (Gill, Murray, Saunders, 2002) brought SQP to large-scale NLP with a sparse implementation. IPOPT (Wächter and Laird, 2006) brought interior point methods to NLP with the filter line search. KNITRO (Byrd, Nocedal, Waltz) offered both SQP and interior point in a single package. Gurobi and CPLEX dominated LP and mixed-integer LP. These solvers are the tools that engineers actually use, and their development required solving problems that pure mathematical theory did not address: numerical stability, warm starting, infeasibility detection, degenerate problems, and scaling.

## Sequential Quadratic Programming

SQP — the dominant algorithm for medium-scale constrained NLP — builds a quadratic model of the Lagrangian at each iteration and solves a constrained quadratic program (QP) to determine the step direction. The QP subproblem has the same constraints as the original NLP (linearized), which makes the step feasible (or nearly so) by construction.

The key insight, due to Wilson (1963) and Han (1977), is that solving the NLP is equivalent to

solving a sequence of QPs, each of which refines the solution. Each QP is fast to solve (quadratic objective, linear constraints), and the sequence converges to the KKT point of the original NLP superlinearly — each iteration roughly squares the error.

SQP's practical advantages: warm-starting (initialize from a nearby previous solution), natural handling of equality constraints, and compatibility with active-set LP solvers for the QP subproblem. Its practical limitation: the QP subproblem requires storing and factoring the Hessian approximation, which costs $O(n^2)$ memory for $n$ variables. For $n > 500$, this becomes impractical on standard hardware.

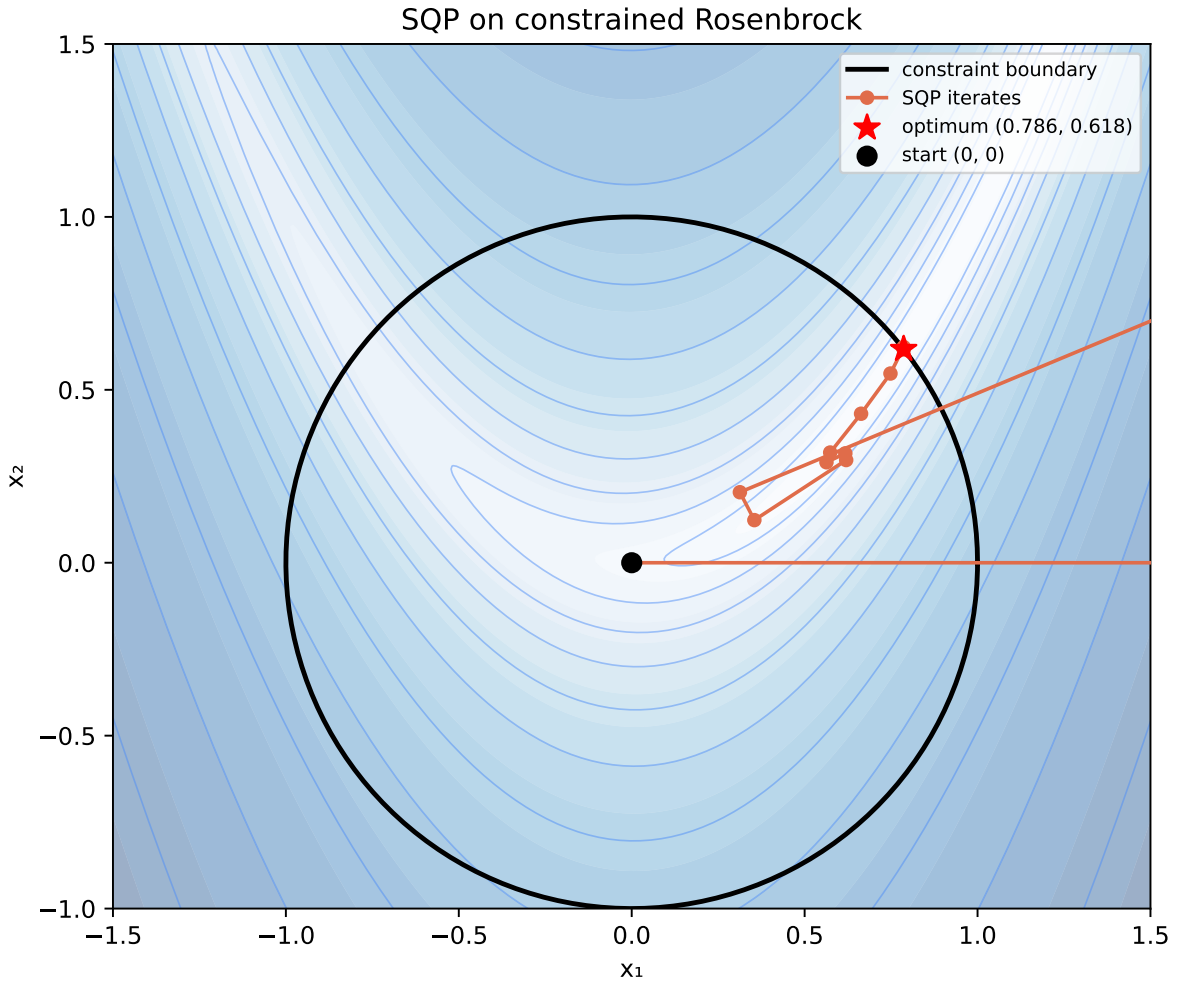


Figure 7: SQP on a constrained NLP: minimize Rosenbrock function subject to a unit disk constraint. Each iteration solves a QP subproblem (shown as the elliptical trust region centered at each iterate). The path converges rapidly to the constrained optimum on the boundary.

## IPOPT and the Filter Line Search

IPOPT (Interior Point OPTimizer), released by Andreas Wächter and Carl Laird in 2006, solved a problem that had blocked practical interior point methods for NLP: globalization.

A pure Newton method applied to the KKT conditions converges quadratically near the solution

but can diverge or oscillate from a distant starting point. Globalization — ensuring convergence from any starting point — traditionally used a merit function: a scalar measure combining objective value and constraint violation, monotonically decreased along the search path. The problem is that choosing the weights in the merit function is nontrivial; wrong weights can stall the method near infeasible regions.

Wächter and Laird's filter line search avoids merit function weights entirely. The filter is a list of (objective value, constraint violation) pairs, maintained as a Pareto front. A new iterate is accepted if it dominates at least one entry in the filter — improves either the objective or the constraint violation compared to some prior iterate. The filter prevents cycling without requiring weight tuning.

IPOPT is now the default NLP solver in Julia's JuMP modeling language, the recommended solver for Pyomo in Python, and the solver used by CasADi (a tool widely used in model predictive control and trajectory optimization). It handles problems with hundreds of thousands of variables and sparse Jacobian structures routinely.

## Putting the History Together

The modern constrained NLP solver is a palimpsest of three centuries of mathematical history. The Lagrangian it maintains descends from Lagrange's 1788 multipliers. The KKT conditions it satisfies were derived by Karush in 1939 and Kuhn and Tucker in 1951. The Newton steps it takes descend from Newton's seventeenth-century method. The quasi-Newton Hessian approximation it builds uses the BFGS formula from 1970. The interior point structure traces to Karmarkar's 1984 paper. The filter line search is Wächter and Laird's 2006 contribution.

No single person designed this machine. It assembled itself over centuries, as different mathematicians solved different pieces of a problem they did not fully realize was the same problem. That is usually how it works.

---

## Summary

The modern optimization toolbox — SQP, IPOPT, KNITRO, Gurobi — represents the engineering work of the 1990s and 2000s that turned theoretical algorithms into robust solvers. SQP handles medium-scale NLP through quadratic subproblems, with warm-starting and natural equality constraint handling. IPOPT handles large-scale sparse NLP through interior point methods with filter line search globalization. Each solver embodies decades of accumulated algorithmic ideas from across the history of the field.

## Further Reading

Wächter and Laird (2006) is the IPOPT paper. Gill, Murray, and Saunders' "SNOPT: An SQP algorithm for large-scale constrained optimization" (*SIAM Review*, 47(1):99–131, 2005) describes the dominant SQP solver. Nocedal and Wright (2006) is the comprehensive algorithmic reference for everything in this chapter.

# The Machine Learns to Optimize

**Learning Objectives**

- Understand backpropagation as automatic differentiation through a computational graph
- See how stochastic gradient descent (Cauchy's algorithm) became the engine of deep learning
- Understand the Adam optimizer and why it dominates neural network training
- Reflect on how optimization connects classical engineering to modern AI

## Cauchy's Algorithm, One More Time

In 2012, Geoffrey Hinton, Ilya Sutskever, and Alex Krizhevsky trained a convolutional neural network called AlexNet using a GPU cluster at the University of Toronto. AlexNet won the ImageNet Large Scale Visual Recognition Challenge by a margin that stunned the computer vision community — it achieved 15.3% top-5 error, compared to 26.2% for the next-best entry. The field changed overnight.

AlexNet was trained using stochastic gradient descent: start at a random point in a space with 60 million dimensions, and iterate $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \hat{\nabla} f(\mathbf{x}_k)$, where $\hat{\nabla} f$ is the gradient estimated from a random mini-batch of training data. This is Augustin-Louis Cauchy's 1847 algorithm, applied to a function with sixty million variables, trained on one million images.

Nothing in Cauchy's paper would have predicted this. The zigzag convergence problem that makes steepest descent slow on ill-conditioned quadratics is also present in neural network training, but it matters less because neural networks are so overparameterized — the loss landscape is so flat in so many directions — that any reasonable descent direction makes progress. The noise introduced by mini-batch sampling turns out to help, not hurt, by preventing the optimizer from settling into sharp, narrow minima that generalize poorly.

## Backpropagation: The Gradient Machine

Training a neural network requires computing the gradient of the loss function with respect to all parameters. For a network with 60 million parameters, computing this gradient by finite differences would require 60 million forward passes — computationally catastrophic.

Backpropagation, popularized by Rumelhart, Hinton, and Williams in their 1986 *Nature* paper (and independently discovered by several others earlier), is the efficient solution. It computes

the exact gradient of any differentiable function defined by a computational graph using the chain rule, in time proportional to the cost of a single forward evaluation.

The key insight: the chain rule can be applied in reverse order, propagating gradient information from the output backward through the network. Each layer computes a local Jacobian; multiplying these Jacobians together gives the full gradient. The total cost is two forward passes (one forward, one backward), regardless of the number of parameters.

Backpropagation is, mathematically, the adjoint method applied to feedforward neural networks. The adjoint method — covered in the optimization lesson series — is the same algorithm applied to partial differential equations and used in aerodynamic shape optimization and solid rocket motor sensitivity analysis. The deep learning community and the computational fluid dynamics community independently developed the same idea for different applications.

## Adam: The Modern SGD

Stochastic gradient descent with a fixed learning rate performs poorly in practice — the learning rate that works early in training (when far from the optimum) is too large late in training (when near the optimum), and vice versa. Adapting the learning rate to the local geometry of the loss function was the key practical advance of the 2010s.

The Adam optimizer (Kingma and Ba, 2015) maintains running estimates of the first and second moments of the gradient:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1)\hat{\nabla} f_k$$
$$v_k = \beta_2 v_{k-1} + (1-\beta_2)\hat{\nabla} f_k^2$$

After bias correction ($\hat{m}_k = m_k/(1-\beta_1^k)$, $\hat{v}_k = v_k/(1-\beta_2^k)$), the update is:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\alpha}{\sqrt{\hat{v}_k} + \epsilon}\hat{m}_k$$

The denominator $\sqrt{\hat{v}_k}$ acts as a per-parameter learning rate scaler: parameters with large recent gradients (likely in a steep direction) get smaller steps; parameters with small recent gradients get larger steps. This is an adaptive preconditioner — a poor man's Newton step that avoids computing the full Hessian.

## Bayesian Optimization: When Evaluations Are Expensive

For problems where the objective function is expensive to evaluate — a neural architecture search requiring days of GPU training, an aerodynamic shape requiring hours of CFD simulation, a propellant formulation requiring synthesis and testing — gradient descent is impractical. The function is a black box; derivatives may be unavailable; and the budget of evaluations is measured in dozens, not millions.

Bayesian optimization treats the objective as a sample from a Gaussian process prior, updates the posterior after each evaluation, and uses an *acquisition function* to decide where to evaluate next. The acquisition function balances exploration (evaluating where uncertainty is high) and exploitation (evaluating where the posterior mean is low). The most common acquisition function is expected improvement (EI): the expected reduction in the best value seen so far.

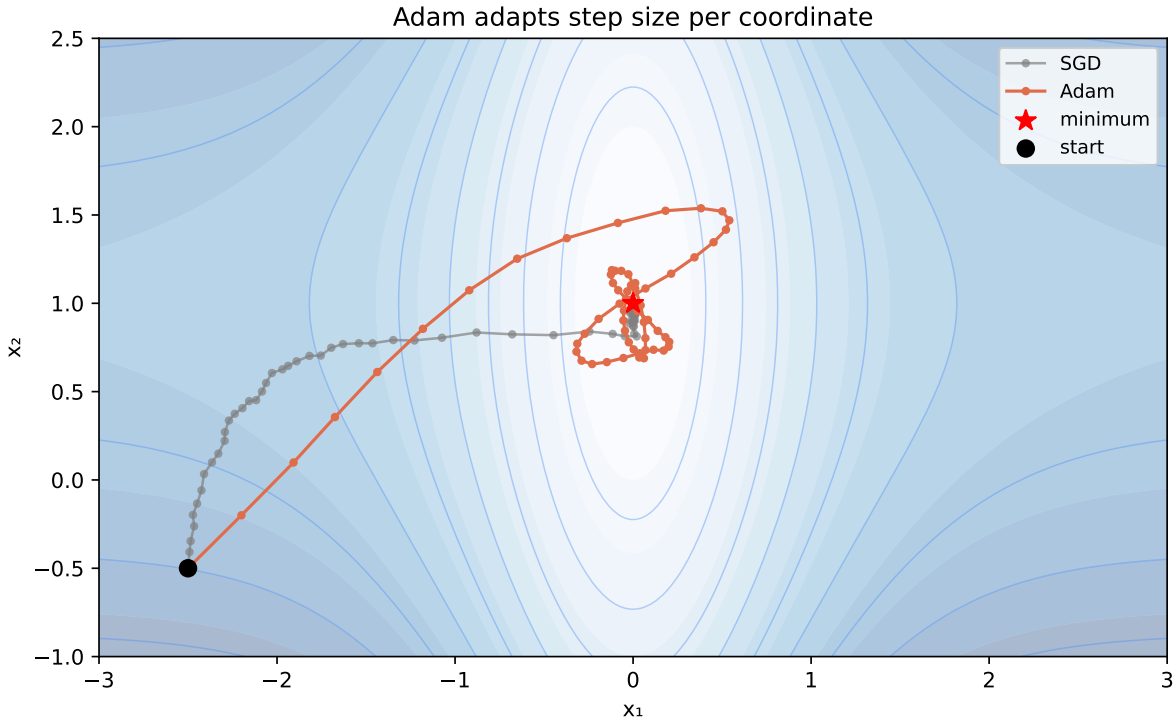


Figure 8: Adam vs. SGD with fixed learning rate on a simple non-convex function. Adam adapts its effective step size per coordinate, navigating the flat plateau more efficiently than fixed-rate SGD. The advantage is most pronounced in high dimensions and near flat regions.

Bayesian optimization is provably sample-efficient for smooth functions, achieving sublinear cumulative regret under mild assumptions on the Gaussian process kernel. In practice, it typically finds good solutions in 50–200 evaluations on problems where gradient methods would need thousands. It is now the standard method for hyperparameter tuning in machine learning.

## The Gradient Runs Through It

Looking back at the three centuries covered by this book, one thread runs through everything: the gradient.

Fermat's stationary points. Euler's first variation. Lagrange's multiplier. Cauchy's steepest descent. Dantzig's reduced costs. Kantorovich's resolving multipliers. The KKT gradient condition. Karmarkar's projective step. IPOPT's Newton iteration. Adam's adaptive gradient. Backpropagation's chain rule.

Every one of these is, at its core, a statement about the gradient of some function — where it vanishes, how to follow it, how to project it, how to modify it, how to combine it with constraint information. The field changes its notation every generation, but the mathematics is recognizably the same mathematics that Cauchy wrote down in two pages in 1847.

The problems change. Lagrange was minimizing the potential energy of a mechanical system. Dantzig was allocating Air Force resources. Hinton was classifying images. The researchers currently using neural networks to fold proteins, predict weather, and design drugs are solving optimization problems that Lagrange could not have imagined. The gradient — his gradient, Cauchy's gradient — is what makes it possible.

---

## Summary

Stochastic gradient descent — Cauchy's 1847 algorithm applied to functions with millions of variables — became the engine of the deep learning revolution. Backpropagation computes exact gradients efficiently through the chain rule applied in reverse, enabling training of networks with billions of parameters. Adam's adaptive step sizing addresses the ill-conditioning problem that afflicts basic gradient descent. Bayesian optimization handles the black-box setting where evaluations are expensive. The thread connecting all of these methods to the history of optimization is the gradient — the fundamental concept introduced by Cauchy and refined by every chapter in between.

## Further Reading

Rumelhart, Hinton, and Williams' "Learning representations by back-propagating errors" (*Nature*, 323:533–536, 1986) is the backpropagation paper. Kingma and Ba's "Adam: A method for stochastic optimization" (ICLR 2015) introduces Adam. Frazier's "A tutorial on Bayesian optimization" (arXiv:1807.02811, 2018) is the best accessible introduction to Bayesian optimization. For the deep learning context generally, Goodfellow, Bengio, and Courville's *Deep Learning* (MIT Press, 2016) is the standard reference; Chapter 8 covers gradient-based optimization in depth.

# References


Cauchy, Augustin-Louis. 1847. "Méthode générale Pour La résolution Des Systèmes d'équations Simultanées." *Comptes Rendus de l'Académie Des Sciences* 25: 536–38.

Dantzig, George B. 1963. *Linear Programming and Extensions.* Princeton University Press.

Goldstine, Herman H. 1980. *A History of the Calculus of Variations from the 17th Through the 19th Century.* Springer.

Kantorovich, Leonid V. 1960. *Mathematical Methods of Organizing and Planning Production.* Management Science.

Karmarkar, Narendra. 1984. "A New Polynomial-Time Algorithm for Linear Programming." *Combinatorica* 4 (4): 373–95.

Kuhn, Harold W., and Albert W. Tucker. 1951. *Nonlinear Programming.* 481–92.

Lagrange, Joseph-Louis. 1788. *Mécanique Analytique.* Desaint.

Nocedal, Jorge, and Stephen J. Wright. 2006. *Numerical Optimization.* 2nd ed. Springer.

Schrijver, Alexander. 1998. *Theory of Linear and Integer Programming.* Wiley.

Wächter, Andreas, and Carl T. Laird. 2006. "On the Implementation of an Interior-Point Filter Line-Search Algorithm for Large-Scale Nonlinear Programming." *Mathematical Programming* 106 (1): 25–57.

# Code Reference

This appendix collects reusable Python functions referenced in the main text. All code uses only the standard scientific Python stack (NumPy, SciPy, Matplotlib) available in the project pixi environment.

### Setup

### Steepest Descent (Chapter 4)

### Lagrange Multiplier Example (Chapter 3)

### Simple LP via Simplex (Chapter 5)

### Adam Optimizer (Chapter 10)