Continuous Optimization (Nonlinear and Linear Programming)

Stephen J. Wright

Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, USA

1 Overview

At the core of any optimization problem is a mathematical model of a system, which could be constructed from physical, economic, behavioral, or statistical principles. The model describes relationships between variables that define the state of the system, and may also place restrictions on the states, in the form of constraints on the variables. The model also includes an objective function, which measures the desirability of a given set of variables. The optimization problem is to find the set of variables that achieves the best possible value of the objective, among all those values that satisfy the constraints.

1.1 Examples

Optimization problems are ubiquitous, as we illustrate with some examples.

- A firm wishes to maximize its profit, given constraints on availability of resources (equipment, labor, raw materials), production costs, and forecast demand.
- In order to forecast weather, we first need to solve a problem to identify the state of the atmosphere a few hours ago. This is done by finding the state that is most consistent with recent meteorological observations (temperature, wind speed, humidity, etc) taken at a variety of locations and times. The model contains differential equations that describe evolution of the atmosphere, statistical elements that describe prior knowledge of the atmospheric state, and an objective that measures the consistency between the atmospheric state and the observations.
- Computer systems for recognizing handwritten digits contain models that read the written character, in the form of a pixellated image, and output their best guess as to the digit that is represented in the image. These models can be "trained" by presenting them with a (typically large) set of images containing known digits. An optimization problem is solved to adjust the parameters in the model so that the error count on the training set is minimized. If the training set is representative of the images that the system will see in future, this optimized model can be trusted to perform reliable digit recognition.
- Given a good that is produced in a number of cities and consumed in other cities, we wish to find the least expensive way to transport the good from supply locations to demand locations. Here, the model consists of a graph that describes the transportation network, capacity constraints, and the cost of transporting one unit of the good between two adjacent locations in the network.
- Given a set of possible investments along with the means, variances, and correlations of their expected returns, an investor wishes to allocate his funds in a way that balances the expected mean return of the portfolio with its variance, in a way that fits his appetite for risk.

These few examples capture some of the wide variety of applications currently seen in the field. As they suggest, the mathematical models that underlie optimization problems vary widely in size, complexity, and structure. They may contain simple algebraic relationships, systems of ordinary or partial differential equations, models derived from Bayesian statistics, and "black-box" models whose internal details are not accessible, and which can be accessed only by supplying inputs and observing outputs.

1.2 Continuous Optimization

In continuous optimization, the variables in the model are nominally allowed to take on a continuous range of values, usually real numbers. This feature distinguishes continuous optimization from discrete or combinatorial optimization, in which the variables may be binary (restricted to the values 0 and 1), integer (for which only integer values are allowed), or more abstract objects drawn from sets with finitely many elements. (Discrete optimization is the subject of another article in this volume.)

Continuous optimization problems are typically solved using algorithms that generate a sequence of values of the variables, known as *iterates*, that converge to a solution of the problem. In deciding how to step from one iterate to the next, the algorithm makes use of knowledge gained at previous iterates, and information about the model at the current iterate, possibly including information about its sensitivity to perturbations in the variables. The continuous nature of the problem allows sensitivities to be defined in terms of first and second derivatives of the functions that define the models.

1.3 Standard Paradigms

Research in continuous optimization tends to be organized into several paradigms, each of which makes certain assumptions on the properties of the objective function, variables, and constraints. To define these paradigms, we group the variables into a real vector x with n components (that is, $x \in \Re^n$), and define the general continuous optimization problem as follows:

$$\min_{x \in \Re^n} f(x) \tag{1a}$$

$$\text{subject to } c_i(x) = 0, \quad i \in \mathcal{E}, \tag{1b}$$

$$c_i(x) \geq 0, \quad i \in \mathcal{I}. \tag{1c}$$

where the objective f and the constraints c_i, $i \in \mathcal{E} \cup \mathcal{I}$ are real-valued functions on $\Re^n$. To this formulation is sometimes added a geometric constraint

$$x \in \Omega, \tag{2}$$

where $\Omega \subset \Re^n$ is a closed convex set. All functions in (1) are assumed to at least be continuous. A point x that satisfies all the constraints is said to be *feasible*.

There is considerable flexibility in the way that a given optimization problem can be formulated, and the choice of formulation can affect the effectiveness with which the problem can be solved. One common reformulation technique is to replace an inequality constraint by an equality constraint plus a bound, by introducing an new "slack" variable:

$$c_i(x) \leq 0 \quad \Leftrightarrow \quad c_i(x) + s_i = 0, \; s_i \geq 0. \tag{3}$$

Referring to the general form (1), we distinguish several popular paradigms.

- In *linear programming*, all objectives and constraints are affine functions of x, that is, they have the form $a^T x + b$, for some $a \in \Re^n$ and $b \in \Re$.

- In *quadratic programming*, we have $f(x) = (1/2)x^T Q x + c^T x + d$, for some $n \times n$ symmetric matrix Q, vector $c \in \Re^n$, and scalar $d \in \Re$; while all constraints c_i are linear. When Q is positive semidefinite, we have a *convex quadratic program.*

- In *convex programming*, the objective f and the negated inequality constraint functions $-c_i$, $i \in \mathcal{I}$ are convex functions, while the equality constraints c_i, $i \in \mathcal{E}$ are affine functions. (These assumptions, along with the convexity and closedness of Ω in the case that (2) is included in the formulation, imply that set of feasible points is closed and convex.)

- *Conic optimization* problems have the form (1), (2), where the set Ω is assumed to be a pointed, closed, convex cone, while the objective f and equality constraints c_i, $i \in \mathcal{E}$ are assumed to be affine. There are no inequalities, that is $\mathcal{I} = \emptyset$.

- In *unconstrained optimization*, the constraints (1b), (1c), and (2) are nonexistent, while the objective f is usually assumed to be smooth, with at least continuous first derivatives. *Nonsmooth optimization* allows f to have discontinuous first derivatives, but it is often assumed that f has some other structure that can be exploited by the algorithms.

- In *nonlinear programming*, the functions f and c_i, $i \in \mathcal{E} \cup \mathcal{I}$ are generally nonlinear but smooth, at least having continuous first partial derivatives on the region of interest.

An important special class of conic optimization problems is *semidefinite programmming*, in which the vector x of unknowns contains the elements of a symmetric $m \times m$ matrix X which is required to be positive semidefinite. It is natural and useful to write this problem in terms of the matrix X as follows:

$$\min_{X \in \mathcal{S}\Re^{m \times m}} C \bullet X \tag{4a}$$

$$\text{subject to } A_i \bullet X = b_i, \ i = 1, 2, \ldots, p; \tag{4b}$$

$$X \succeq 0, \tag{4c}$$

Here, $\mathcal{S}\Re^{m \times m}$ denotes the set of symmetric $m \times m$ matrices, the matrices C and A_i, $i = 1, 2, \ldots, p$ all belong to $\mathcal{S}\Re^{m \times m}$, while the operator $\bullet$ is defined on pairs of matrices in $\mathcal{S}\Re^{m \times m}$ as follows: $A \bullet B = \sum_{i=1}^{m} \sum_{j=1}^{m} A_{ij} B_{ij} = \text{trace}(AB)$. The constraint (4c) instantiates the geometric constraint (2).

Terminology. "Mathematical programming" is a historical term that encompasses optimization and such closely related areas as complementarity problems. Its origins date to the 1940s, with the development of the simplex method of George Dantzig, the first effective method for linear programming (8). The term "programming" referred originally to the formalized, systematic mathematical procedure by which problems can be solved. Only later did "programming" become roughly synonymous with "computer programming," causing some slight confusion which optimization reserachers have often been called on to explain. The more modern term "optimization" is generally preferred, although the term "programming" is still attached (probably forever) to such problems as linear programming and integer programming.

1.4 Scope of Research

Research in optimization encompasses study of the mathematical properties of the problems themselves; development, testing, and analysis of algorithms for solving particular classes of problems (such as one of the paradigms described above); and development of models and algorithms for specific application areas. We give a brief description of each of these aspects.

One topic of fundamental interest is the characterization of solution sets: Are there verifiable conditions that we can check to determine whether a given point is a solution to the optimization problem? Given the uncertainty that is present in many practical settings, we may also be interested in the sensitivity of the solution to perturbations in the data or in the objective and constraint functions. Ill-conditioned problems are those in which the solution can change significantly when the data or functions change slightly. Another important fundamental concept is *duality*: Often, the data and functions that define an optimization problem can be rearranged to produce a new "dual" problem that is related to the original problem in interesting ways. The concept of duality can also be of great practical importance in designing more efficient formulations and algorithms.

The study of algorithms for optimization problems blends theory and practice. The design of algorithms that work well on practical problems requires a good deal of intuition and testing. Most algorithms in use today have a solid theoretical basis, but the theory often allows wide latitude in the choice of certain parameters, and algorithms are often "engineered" to find suitable values for these parameters and to incorporate other heuristics. Analysis of algorithms tackles such issues as whether the iterates can be guaranteed to converge to a solution (or some other point of interest); whether there is an upper bound on the number of iterations needed, as a function of the size or complexity of the problem; and the rate of convergence, particularly after the iterates enter a certain neighborhood of the solution. Algorithmic analysis is typically worst-case in nature. It gives important indications about how the algorithm will behave in practice, but does not tell the whole story. Famously, the simplex method is known to perform badly in the worst case — its running time may be exponential in the problem size — yet its performance on most practical problems is impressively good.

Development of software that implements efficient algorithms is another important activity. High-quality codes are available both commercially and in the public domain. Modeling tools — high-level languages that serve as a front-end to algorithmic software packages — have become more popular in recent years. They relieve the user of much of the burden of transforming their practical problem to a set of functions (the objective and constraints in (1)), allowing the model to be expressed in intuitive terms, closer to the application.

With the growth in the size and complexity of practical optimization problems, issues of modeling, formulation, and customized algorithm design have become more prominent. A particular application can be formulated as an optimization problem in many different ways, and different formulations can lead to very different solver performance. Experience and testing is often required to identify the most effective formulation.

Many modern applications cannot be solved effectively with packaged software for one of the standard paradigms of Section 1.3. It is necessary to assemble a customized algorithm, drawing on a variety of algorithmic elements from the optimization toolbox, and also on tools from other disciplines in scientific computing. This approach allows the particular structure or context of the problem to be exploited. Examples of special context include the following. Low-accuracy solutions may suffice for some problems. Algorithms that require less data movement, or sampling from a large data set, or the ability to handle streaming data, may be essential in other settings. Algorithms that produce (possibly suboptimal) solutions in real time may be essential in such contexts as industrial control.

1.5 Connections

Continuous optimization is a highly interconnected discipline, having close relationships to other areas of mathematics, with scientific computing, and with numerous application areas. It also has close connections to discrete optimization, which often requires continuous optimization problems to be solved as subproblems or relaxations.

In mathematics, continuous optimization relies heaving on various forms of mathematical analysis, especially real analysis and functional analysis. Certain types of analysis have been developed in close association the discipline of optimization, including convex analysis [22], nonsmooth analysis [6], and variational analysis [23]. The theory of computational complexity also plays a role in the study of algorithms. Game theory is particularly relevant when we examine duality and optimality conditions for optimization problems. Control theory is also relevant, for framing problems involving dynamical models and as an important source of applications for optimization. Statistics provides vital tools for stochastic optimization and for optimization in machine learning, in which the model is available only through sampling from a data set.

Continuous optimization also intersects with many areas in numerical analysis and scientific computing. Numerical linear algebra is vitally important, since many optimization algorithms generate a sequence of linear approximations, which must be solved with linear algebra tools. Differential equation solvers are important counterparts to optimization in such applications as data assimilation and distributed parameter identification, which involve optimization of ODE and PDE models. The ubiquity of multicore architectures and the wide availability of cluster computing has given new prominence to parallel algorithms in some areas (such as machine learning), requiring the use of software tools for parallel computing.

Finally, we mention some of the many connections between optimization and several application areas within which it has become deeply embedded. Machine learning uses optimization algorithms extensively, to perform classification and learning tasks. The challenges posed by machine learning applications (for example, large data sets) have been driving recent developments in stochastic optimization and large-scale unconstrained optimization. Compressed sensing, in which sparse signals are recovered from randomized encodings, also relies heavily on optimization formulations and specialized algorithms. Engineering control is a rich source of challenging optimization problems at many scales, frequently involving dynamic models of plant processes. In these and many other areas, practitioners have made important contributions to all aspects of continuous optimization.

2 Basic Principles

We mention here some basic theory that underpins continuous optimization, and that serves as a starting point for the algorithms outlined in later sections.

Possibly the most fundamental issues are: How do we define a solution to the problem, and how do we recognize such a point? The answers become more complicated as we expand the classes of functions allowed in the formulation. The type of solution most amenable to analysis is a *local solution.* The point x^* is a local solution for (1) if x^* is feasible, and there is an open neighborhood $\mathcal{N}$ of x^* such that $f(x) \geq f(x^*)$ for all feasible points $x \in \mathcal{N}$. Further, x^* is a *strict local solution* if $f(x) > f(x^*)$ for all feasible $x \in \mathcal{N}$ with $x \neq x^*$. A *global solution* is a point x^* such that $f(x) \geq f(x^*)$ for *all* feasible x.

As we see below, we can use the derivatives of the objective and constraint functions to construct testable conditions that verify that x^* is a local solution, under certain assumptions. It is difficult to verify global optimality, even when the objective and constraints are smooth, because of the difficulty of gaining a global perspective on these functions. However, in *convex optimization*, where the objective f is a convex function and the set of feasible points is also convex, all local solutions are global solutions. (Convex optimization includes linear programming and conic optimization as special cases.)

Global optimization techniques have also been devised for certain classes of nonconvex problems. It is possible to prove results about the performance of such methods when the function f satisfies additional properties (such as Lipschitz continuity, with known Lipschitz constant) and the feasible region is bounded. One class of methods for solving the global optimization problem uses a process of subdividing the feasible region and using information about f to obtain a lower bound on the objective in that region, leading to a branch-and-bound algorithm akin to methods used in integer programming.

We turn now to characterizations of local solutions for problems defined by smooth functions, assuming for simplicity that f and c_i, $i \in \mathcal{E} \cup \mathcal{I}$ have continuous second partial derivatives. We use $\nabla f(x)$ to denote the gradient of f (the vector in $\Re^n$ of first partial derivatives) and $\nabla^2 f(x)$ to denote the Hessian of f (the $n \times n$ matrix of second partial derivatives). An important tool, both in the characterization of solutions for smooth problems and in the design of algorithms, is *Taylor's Theorem.* This result can be used to estimate the value of f by using its derivative information at a nearby point. For example, we have

$$f(x+p) = f(x) + \nabla f(x)^T p + o(\|p\|), \tag{5a}$$

$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x) p + o(\|p\|^2), \tag{5b}$$

where the notation $o(t)$ indicates a quantity that goes to zero faster than t. These formulae can be used to construct low-order approximations to the problem (1) that are valid in the neighborhood of a current iterate x, and can thus be used to identify a possibly improved iterate $x + p$.

For unconstrained optimization of a smooth function f, we have the following necessary condition.

If x^* is a local solution of $\min_x f(x)$, then $\nabla f(x^*) = 0$.

Note that this is only a *necessary* condition; it is possible to have $\nabla f(x) = 0$ without x being a minimizer. (An example is the scalar function $f(x) = x^3$, which has no minimizer but which has $\nabla f(0) = 0$.) To complement this result, we have the following sufficient condition.

If we have a point x^* such that $\nabla f(x^*) = 0$ with $\nabla^2 f(x^*)$ positive definite, then x^* is a strict local solution of $\min_x f(x)$.

Turning to constrained optimization — the general form (1), with smooth functions — identification of local solutions becomes somewhat more complex. We can obtain a necessary condition based on the gradients of ∇f and ∇c_i, but this depends on an additional condition called a *constraint qualification*, which ensures that the linear approximation to the feasible set, based on the linear approximations (5a) to the constraint functions c_i around the point x, capture the true geometry of the feasible set near x.

A central role in characterizing solutions of constrained optimization problems is played by the *Lagrangian* function, defined as follows:

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x). \tag{6}$$

This is a linear combination of objective and constraints, where the weights λ_i are called *Lagrange multipliers*. At a local solution x^* for (1), the following conditions will hold for some values of λ_i^*, $i \in \mathcal{E} \cup \mathcal{I}$:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \tag{7a}$$
$$c_i(x^*) = 0, \quad i \in \mathcal{E}, \tag{7b}$$
$$c_i(x^*) \geq 0, \quad i \in \mathcal{I}, \tag{7c}$$
$$\lambda_i^* \geq 0, \quad i \in \mathcal{I}, \tag{7d}$$
$$\lambda_i^* c_i(x^*) = 0, \quad i \in \mathcal{I}. \tag{7e}$$

Condition (7e) is a *complementarity condition* that indicates complementarity between each inequality constraint value $c_i(x^*)$ and its Lagrange multiplier λ_i^*: For each i, at least one of these two quantities must be zero. Roughly speaking, the Lagrange multipliers measure the sensitivity of the optimal objective value $f(x^*)$ to perturbations in the constraints c_i. The conditions (7) are often known as the *Karush-Kuhn-Tucker conditions* after their inventors, or *KKT conditions* for short.

When the functions in (1) are nonsmooth, it becomes harder to define optimality conditions, as even the concept of derivative becomes more complicated. We consider the simplest problem instance of this type — the unconstrained problem $\min_x f(x)$, where f is a convex (possibly nonsmooth) function. The *subdifferential* of f at a point x is defined from the collection of supporting hyperplanes to f at x:

$$\partial f(x) := \{v \mid f(z) \geq f(x) + v^T(z - x) \\ \text{for all } z \text{ in the domain of } f\}.$$

For example, the function $f(x) = \|x\|_1 = \sum_{i=1}^n |x_i|$ is nonsmooth, with subdifferential consisting of the vectors v such that

$$v_i \begin{cases} = +1 & \text{if } x_i > 0 \\ \in [-1, 1] & \text{if } x_i = 0 \\ = -1 & \text{if } x_i < 0. \end{cases}$$

When f is smooth at x in addition to being convex, we have $\partial f(x) = \{\nabla f(x)\}$. A necessary and sufficient condition for x^* to be a solution of $\min_x f(x)$ is that $0 \in \partial f(x^*)$.

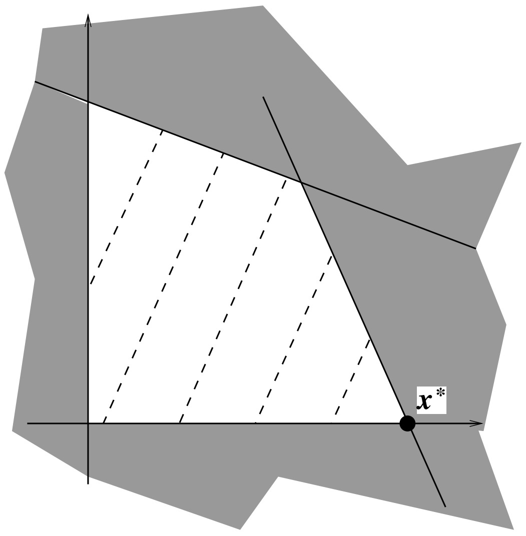

Figure 1: Feasible region (unshaded), objective function contours (dashed lines), and optimal vertex x^* for a linear program in two variables.

3 Linear Programming

Consider the problem

$$\min_x \; c^T x \; \text{ subject to } \; Ax = b, \; x \geq 0, \tag{8}$$

where $x \in \Re^n$ as before, $b \in \Re^m$ is the right-hand side, and $A \in \Re^{m \times n}$ is the constraint matrix. Any optimization problem with an affine objective function and affine constraints can be written in this standard form, after some elementary transformations. As illustrated by the example in Figure 1, the feasible region for the problem (8) is polyhedral, and the countours of the objective function are lines.

There are three possible outcomes for a linear program:

(a) The problem is *infeasible*, that is, there is no point x that satisfies $Ax = b$ and $x \geq 0$;

(b) The problem is *unbounded*, that is, there is a sequence of feasible points x^k such that $c^T x^k \downarrow -\infty$;

(c) The problem has a solution, that is, there is a feasible point x^* such that $c^T x^* \leq c^T x$ for all feasible x.

When a solution exists (case (c)) it may not be uniquely defined. However, we can note that the set of solutions itself forms a polyhedron, and that at least one solution lies at a *vertex* of the feasible set, that is, a point which does not lie in the interior of a line joining any other two feasible points.

By rearranging the data in (8), we obtain another linear program called the *dual*:

$$\max_{\lambda,s} b^T\lambda \text{ subject to } A^T\lambda + s = c, \; s \geq 0. \quad (9)$$

(In discussions of duality, the original problem (8) is called the *primal* problem.) The primal and dual problems are related by a powerful *duality theory* which has important practical implications. *Weak duality* states that if x is a feasible point for (8) and (λ, s) is a feasible point for (9), then the primal objective is greater than or equal to the dual objective. This statement is easily proved in a single line:

$$c^T x = (A^T\lambda + s)^T x \geq \lambda^T A x = \lambda^T b.$$

The other fundamental duality result — *strong duality* — states that there are three possible outcomes for the pair of problems (8) and (9):

(a) One of the two problems is infeasible and the other is unbounded;

(b) Both problems are infeasible;

(c) (8) has a solution x^* and (9) has a solution (λ^*, s^*) with objective functions equal: $c^T x^* = b^T \lambda^*$.

Specializing (7), we see that the primal and dual problems share a common set of KKT conditions:

$$Ax = b, \;\; A^T\lambda + s = c, \quad (10a)$$
$$x \geq 0, \;\; s \geq 0, \quad (10b)$$
$$x_i s_i = 0, \;\; i = 1, 2, \ldots, n. \quad (10c)$$

If (x^*, λ^*, s^*) is any vector triple that satisfies these conditions, x^* is a solution of (8) and (λ^*, s^*) is a solution of (9).

We now discuss the two most important classes of algorithms for linear programming.

3.1 Simplex Method

The simplex method, devised by George Dantzig in the late 1940s (and described in [9]) remains a fundamental approach of practical and theoretical importance in linear programming. Geometrically speaking, the simplex method moves from vertex to neighboring vertex of the feasible set, decreasing the objective function with each move, and terminating when it cannot find a neighboring vertex with a lower objective value. The method is implemented by maintaining a *basis* — a subset of m out of the n components of x that are allowed to be nonzero at the current iteration. The values of these basic components of x are determined uniquely by the m linear constraints $Ax = b$. Each step of the simplex method starts by choosing a non-basic variable to enter the basis. This variable is allowed to increase away from zero, a process which, because of the requirement to maintain feasibility of the linear constraints $Ax = b$, causes the values of the existing basic variables to change. The entering variable is allowed to increase to the point where one of the basic variables reaches zero, upon which it leaves the basis, and the iteration is complete.

Efficient implementation of the simplex method depends both on good “pricing” strategies, to choose which nonbasic variable should enter the basis, and efficient linear algebra, to update the values of the basic variables as the entering variable increases away from zero. Both topics have seen continued development over the years, and highly effective software is available, both commercially and in the public domain. Specialized, highly efficient versions of the simplex method exist for some special cases of linear programming, such as those arising from transportation or routing over networks.

The simplex method is an example of an active-set method: It maintains a subset of the inequality constraints that is held to be *active* (that is, enforced at equality) at each iteration, and changes this set only slightly from one iteration to the next. (In the problem (8), the inequality constraints are the bounds $x_i \geq 0$, $i = 1, 2, \ldots, n$, so the active set at iteration k is the complement of the basis at x^k.)

The theoretical properties of the simplex method remain a source of fascination because, despite its practical efficiency, its worst-case behavior is poor. A famous example [14] shows that the number of steps may be exponential in the dimension of the problem. There have been various attempts to understand the "average case" behavior, in which the number of iterations required is roughly linear in the problem dimensions. The "smoothed analysis" of Spielman and Teng [24] shows that small perturbations in the data of a problem for which simplex behaves badly yields a problem that requires only polynomially many iterations.

An algorithm with polynomial complexity (in the worst case) was announced in 1979: Khachiyan's ellipsoid algorithm [13]. Though of great theoretical interest, it was not a practical alternative to simplex. The interior-point revolution began with Karmarkar's algorithm [12], also a polynomial-time approach. This method had much better computational properties than the ellipsoid approach. It motivated a new class of algorithms — primal-dual interior-point methods — which not only had attractive theoretical properties, but were also truly competitive with simplex on practical problems. We describe these next.

3.2 Interior-Point Methods

As their name suggests, primal-dual interior-point methods generate a sequence of iterates (x^k, λ^k, s^k), $k = 1, 2, \ldots$ in both primal and dual variables, in which x^k and s^k contain all positive numbers (that is, they are strictly feasible with respect to the constraints ($x \geq 0$ and $s \geq 0$ in (10b)). Steps between iterates are obtained by applying Newton's method to a perturbed form of the condition (10c) in which the right-hand side 0 is replaced by a positive quantity $\mu_k > 0$, which is gradually decreased to zero as $k \to \infty$. The Newton equations for each step $(\Delta x^k, \Delta\lambda^k, \Delta s^k)$ are obtained from a linearization of these perturbed KKT conditions, specifically,

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta\lambda^k \\ \Delta s^k \end{bmatrix} = -\begin{bmatrix} A^T\lambda^k + s^k - c \\ Ax^k - b \\ X^k S^k e - \mu_k \mathbf{1} \end{bmatrix},$$

where X^k is the diagonal matrix whose diagonal elements come from x^k, S^k is defined similarly, and $\mathbf{1}$ is the vector of length n whose elements are all 1. The new iteration is obtained by setting

$$\begin{aligned}(x^{k+1}, &\lambda^{k+1}, s^{k+1}) \\ &= (x^k + \alpha_k \Delta x^k, \lambda^k + \beta_k \Delta\lambda^k, s^k + \beta_k \Delta s^k),\end{aligned}$$

where α_k and β_k are steplengths in the range $[0, 1]$ chosen so as to ensure that $x^{k+1} > 0$ and $s^{k+1} > 0$, among other goals. Convergence, with polynomial complexity, can be demonstrated under appropriate schemes for choosing μ_k and the steplengths α_k and β_k. Clever schemes for choosing these parameters and for enhancing the search directions using "second-order corrections" lead to good practical behavior. See [27] and the many references therein for further details.

Primal-dual interior-point methods have the additional virtue that they are easily extendible to convex quadratic programming and montone linear complementarity problems, with only minor changes to the algorithm and the convergence theory.

4 Unconstrained Optimization

Consider the problem of simply minimizing a function without constraints:

$$\min_x f(x), \tag{11}$$

where f has at least continuous first derivatives. This problem is important in its own right. It also appears as a subproblem in many methods for constrained optimization, and serves to illustrate several algorithmic techniques that can be applied also to the constrained case.

4.1 First-Order Methods

The Taylor approximation (5a) shows that f decreases most rapidly in the direction of the negative gradient vector $-\nabla f(x)$. Steepest descent methods move in this direction, each iteration having the form

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k), \tag{12}$$

for some positive steplength α_k. A suitable value of α_k can be found by performing (approximately) a one-dimensional search along the durection $-\nabla f(x^k)$, thus guaranteeing a decrease in f at every iteration. In some When further information about f is available, it may be possible to choose α_k to guarantee descent in f without doing a line search. An nonstandard approach [1] chooses α_k by a formula that allows f to increase (sometimes dramatically) on some iterations, while often achieving better long-term behavior than standard steepest-descent approaches.

For the case of convex f, there has been renewed attention to accelerated first-order methods that still require only the calculation of a gradient ∇f at each step, but that have more attractive convergence rates than steepest descent, both in theory and in practice. The common aspect of these methods is a "momentum" device, in which the step from x^k to x^{k+1} is based not just on the latest gradient $\nabla f(x^k)$ but also on the step from the previous iterate x^{k-1} to the current iterate x^k. In *heavy-ball* and *conjugate gradient* methods, the steps have the form

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k) + \beta_k (x^k - x^{k-1}),$$

for positive parameters α_k and β_k that are motivated and implemented in a variety of ways. Other accelerated methods described by Nesterov [18] (see also [2]) retain this spirit, but differ in the derivation of the momentum term. Rather than generating a single sequence of iterates $\{x^k\}$, these methods produce two interleaved series of iterates.

For convex f, first-order methods are characterized in some cases by linear convergence (with the error in x^k decreasing to zero in a geometric sequence) or sublinear convergence (with the error decreasing to zero but not geometrically, typically at a rate of $1/k$ or $1/k^2$, where k is the iteration number).

4.2 Superlinear Methods

When second derivatives of f are available, we can use the second-order Taylor approximation (5b) to motivate *Newton's method*, a fundamental algorithm in both optimization and nonlinear equations. When $\nabla^2 f$ is positive definite at the current iterate x^k, we can define the step p^k to be the minimizer of the right-hand side of (5b) (with the $o(\|p\|^3)$ term omitted), yielding the formula

$$p^k = -[\nabla^2 f(x^k)]^{-1} \nabla f(x^k). \tag{13}$$

The next iterate is defined by choosing a steplength $\alpha_k > 0$ and setting $x^{k+1} = x^k + \alpha_k p^k$. This method is characterized by *quadratic* convergence, in which the error in x^{k+1} is bounded by the *square* of the error in x^k, for all k sufficiently large. (The number of correct digits in x^k doubles on each of the last few iterations.)

Enhancements of the basic approach based on (13) yield more robust and general implementations. For example, the Hessian matrix $\nabla^2 f(x^k)$ may be modified during computation of p^k to ensure that it is a descent direction for f. Another important class of methods known as *quasi-Newton* methods avoids the calculation of second derivatives altogether, instead replacing $\nabla^2 f(x^k)$ in (13) by an approximation B_k that is constructed using first derivative information. The possibility of such an approximation is a consequence of another form of Taylor's theorem, which posits the following relationship between two successive gradients:

$$\nabla f(x^{k+1}) - \nabla f(x^k) \approx \nabla^2 f(x^k)(x^{k+1} - x^k).$$

In updating the Hessian approximation to B_{k+1} after the step to x^{k+1} is taken, we ensure that B_{k+1} mimics this property of the true Hessian, that is, we enforce the condition

$$\nabla f(x^{k+1}) - \nabla f(x^k) \approx B_{k+1}(x^{k+1} - x^k).$$

We obtain a variety of quasi-Newton methods by imposing various other conditions on B_{k+1}, for example, closeness to B_k in some metric, and positive semidefiniteness. *Limited memory* quasi-Newton methods [15] store B_k implicitly by means of the difference vectors between successive iterates and successive gradients at a limited number of prior iterations — typically between 5 and 20.

4.3 Derivative-Free Methods

Methods that require the users to supply only function values f (and not gradients or Hessians) have been enormously popular over many years. More recently, they have attracted the attention of optimization researchers who have tried to improve their performance, equip them with a convergence theory and customize them to certain specific classes of problems, such as problems in which f is obtained from a simulation.

In the absence of gradient or Hessian values, it is sometimes feasible to use finite differencing to construct approximations to these higher-order quantities, and then apply the methods described above. Another possible option is to use algorithmic differentiation [10] to obtain derivatives directly from computer code and, once again, use them in the algorithms described above.

Methods that use only function values are usually suited best to problems of modest dimension n. *Model-based methods* use interpolation among function values at recently visited points to construct a model of the function f. This model is used to generate a new candidate, which is accepted as the next iterate if it yields a sufficient improvement in the function value over the best point found so far. The model is updated by changing the set of points on which the interpolatiion is based, replacing older points with higher values of f by newer points with lower function values. *Pattern-search methods* take candidate steps along a certain frame of directions, shrinking step lengths as needed to evaluate a new iterate with a lower function value. After a successful step, the step length may be increased, to speed future progress. Appropriate maintenance of the set of search directions is crucial to efficient implementation and valid convergence theory. Another derivative-free method is the enormously popular *simplex method* of Nelder and Mead [16]. This method — unrelated to the method of the same name for linear programming — maintains a set of $n + 1$ points that form the vertex of a simplex in $\Re^n$. At each iteration, it replaces one of these points with a new one, by expanding or contracting the simplex along promising directions or reflecting one of the vertices through its opposite face. Various attempts have been made in the years since to improve the performance of this method and to develop a convergence thory.

A recent book [8] provides more detail on these and other derivative-free methods.

4.4 Stochastic Gradient Methods

Important problems have been identified recently for which evaluation of ∇f or even f is computationally expensive, but it is possible to obtain an unbiased estimate of ∇f cheaply. Such problems are common in data analysis, where f typically has the form

$$f(x) = (1/N) \sum_{i=1}^{N} f_i(x),$$

for large N, where each f_i depends on a single item in the data set. If i is selected at random from $\{1, 2, \ldots, N\}$, the vector $g^k = \nabla f_i(x^k)$ is an unbiased estimate of $\nabla f(x^k)$. For convex f, methods that use this approximate gradient information have been a focus of work in the optimization and machine learning communities for some years (see for example [17]), and efforts have recently intensified as their wide applicablity has become evident. The basic iteration has the form $x^{k+1} = x^k - \alpha_k g^k$, where the choice of g^k may be based on additional information about f, such as lower and upper bounds on its curvature. (Line searches are not pratical in this setting, as evaluation of f is assumed to be too expensive.) Additional devices such as averaging of the iterates x^k or the gradient estimates g^k enhance the properties of the method in some settings, such as when f is only weakly convex. Typical convergence analysis shows that the *expected* value of the error in x^k, of the difference between the function value after k iterations and its optimal value, approach zero at a sublinear rate, like $1/k$ or $1/\sqrt{k}$.

5 Conic Optimization

Conic optimization problems have the form

$$\min\, c^T x \;\text{ subject to }\; Ax = b,\; x \in \Omega, \tag{14}$$

where Ω is closed, convex cone. They include linear programming (8) and semidefinite programming (4) as special cases. It is possible to design generic algorithms with good complexity properties for this problem class provided that we can identify a certain type of *barrier function* for Ω. A barrier function φ is convex with domain the interior of Ω, with $\varphi(x) \to \infty$ as x approaches the boundary of Ω. The additional property required for an efficient algorithm is *self-concordancy*, which is the property that for any $x \in \operatorname{dom} \varphi$ and any v, we have

$$|\varphi'''(x)vvv| \le 2[\varphi''(x)vv]^{3/2}.$$

Because the third derivatives are bounded in terms of the second derivatives, the function φ is well approximated (locally at least) by a quadratic, so we can derive complexity bounds on Newton's method applied to φ, with a suitable steplength scheme. We can use this barrier function to define a interior-point method in which each iterate x^k obtained by finding an approximate minimizer of the following equality-constrained optimization problem:

$$\min_x \; c^T x + \mu_k \varphi(x) \;\text{ subject to }\; Ax = b, \qquad (15)$$

where the positive parameter μ_k can be decreased gradually to zero as k increases, as in interior-point methods for linear programming. One or more steps of Newton's method can be used to find the approximate solution to the subproblem (15), starting from the previous iterate.

For linear programming the cone $\Omega = \{x \,|\, x \ge 0\}$ admits a self-concordant barrier function $\varphi(x) = -\sum_{i=1}^n \log x_i$. In semidefinite programming, where Ω is the cone of positive semidefinite matrices, we have $\varphi(X) = -\log\det X$.

The most successful interior-point methods for semidefinite programming in practice are primal-dual methods rather than primal methods. These are (nontrivial) extensions of the linear programming approaches of Subsection 3.2; see [25] for a description.

See [19, 5] for more on algorithms for convex and conic optimization.

6 Nonlinear Programming

We turn next to methods for nonlinear programming, in which the functions f and c_i in (1) are smooth nonlinear functions. A basic principle used in constructing algorithms for this problem is successive approximation of the nonlinear program by simpler problems, such as quadratic programming or unconstrained optimization, to which methods from the previous sections can be applied. Taylor's theorem is instrumental in constructing these approximations, using first- or second-order expansions of functions around the current iterate x^k, and possibly also the current estimates of the Lagrange multipliers for the constraints (1b) and (1c). The optimality conditions described in Section 2 also play a central role in algorithm design. Further information about algorithms for nonlinear programming can be found in [20, 4].

6.1 Gradient Projection

Gradient projection is an extension of the steepest descent approach for unconstrained optimization, in which steps are taken along the negative gradient direction but projected onto the feasible set. Considering the formulation

$$\min \; f(x) \;\text{ s.t. }\; x \in \Omega,$$

the basic gradient projection step is

$$x^{k+1} = P_\Omega(x^k - \alpha_k \nabla f(x^k)),$$

where $P_\Omega(\cdot)$ denotes projection onto the closed convex constraint set Ω. This approach may be practical if the projection can be computed inexpensively, as is the case when Ω is a "box" defined by bounds on the variables. It is possible to enhance the gradient method by using second-order information in a selective way (simple projection of the Newton step does not work). See [4] for additional details.

6.2 Sequential Quadratic Programming

In sequential quadratic programming (SQP), we use Taylor's theorem to form the following aprpoximation of (1) around the current point x^k:

$$\begin{aligned} &\min_{d \in \Re^n} \; \nabla f(x^k)^T d + \frac{1}{2} d^T H_k d && (16a)\\ &\text{s.t. } \; c_i(x^k) + \nabla c_i(x^k)^T d = 0, \quad i \in \mathcal{E}, && (16b)\\ &\qquad c_i(x^k) + \nabla c_i(x^k)^T d \le 0, \quad i \in \mathcal{I}, && (16c) \end{aligned}$$

where H_k is a symmetric matrix. Denoting the solution of (16) by d^k, the next iterate is obtained by setting

$$x^{k+1} = x^k + \alpha_k d^k, \qquad (17)$$

for some step length $\alpha_k > 0$. The problem (16) is a quadratic program; it can be solved with methods of active-set or interior-point type. The matrix H_k may contain second-order information from both objective and constraints; an "ideal" value is the Hessian of the Lagrangian function defined in (6), that is, $H_k = \nabla^2_{xx}\mathcal{L}(x^k, \lambda^k)$, where λ^k are estimates of the Lagrange multipliers, obtained for example from the solution of the subproblem (16) at the previous iteration. When second derivatives are not readily available, H_k could be a quasi-Newton approximation to the Lagrangian Hessian, updated by formulae similar to those used in unconstrained optimization.

A line search can be performed to find a suitable value of α_k in (17). An alternative approach to stabilizing SQP is to add a "trust region" to the subproblem (16), in the form of a constraint $\|d\|_\infty \le \Delta_k$, for some $\Delta_k > 0$.

6.3 Interior-Point Methods

The interior-point methods for linear programming described in Subsection 3.2 can be extended to nonlinear programming, and software based on such extensions has been highly successful. To avoid notational clutter, we consider a formulation of nonlinear programming containing nonnegativity constraints on x along with equality constraints:

$$\min\ f(x)\ \text{ s.t. }\ c_j(x) = 0,\ j \in \mathcal{E};\ \ x \ge 0. \tag{18}$$

(This problem is no less general than (1); simple transformations can be used to express (1) in the form (18).) Following (7), and introducing an additional vector s in the style of (10), we write the optimality conditions for this problem as follows:

$$\nabla f(x) - \sum_{j\in\mathcal{E}} \lambda \nabla c_j(x) - s = 0, \tag{19a}$$

$$c_j(x) = 0,\ j \in \mathcal{E}, \tag{19b}$$

$$x \ge 0,\ s \ge 0, \tag{19c}$$

$$x_i s_i = 0,\ i = 1, 2, \ldots, n. \tag{19d}$$

As in linear programming, interior-point methods generate a sequence of iterates (x^k, λ^k, s^k) in which all components of x^k and s^k are strictly positive. The basic primal-dual step is obtained by applying Newton's method at (x^k, λ^k, s^k) to the nonlinear equations defined by (19a), (19b), and (19d), with the right-hand side in (19d) replace by a positive parameter μ_k, which is reduced to zero gradually at the iterations progress. The basic approach can be enhanced in various ways: quasi-Newton approximations, line searches or trust regions, second-order corrections to the search direction, and so on. A description of a successful interior-point code can be found in [26].

6.4 Augmented Lagrangian

An approach for solving (18), first proposed in the early 1970s, is enjoying renewed popularity because of its successful use in new application areas. Originally known as the "method of multipliers," it is founded on the following augmented Lagrangian function:

$$\mathcal{L}_A(x, \lambda; \mu) := f(x) + \sum_{j\in\mathcal{E}} \lambda_j c_j(x) + \frac{1}{2\mu} \sum_{j\in\mathcal{E}} c_j^2(x), \tag{20}$$

for some positive parameter μ. The method defines a sequence of primal-dual iterates (x^k, λ^k) for a given sequence of parameters $\{\mu_k\}$, where each iteration is defined as follows:

- Obtain x^{k+1} by solving (approximately) the problem

$$\min_x\ \mathcal{L}_A(x.\lambda^k; \mu_k)\ \text{ s.t. } x \ge 0; \tag{21}$$

- Update Lagrange multipliers:

$$\lambda_j^{k+1} = \lambda_j^k + c_j(x^k)/\mu_k,\ \ j \in \mathcal{E};$$

- Choose $\mu_{k+1} \in (0, \mu_k]$ by some heuristic.

The method replaces the original nonlinearly constrained problem with a sequence of bound-constrained problems (21). Unlike in interior-point methods, it is not necessary to drive the parameters μ_k to zero to obtain satisfactory convergence. Although the motivation for this approach is perhaps not as clear as for other algorithms, it can be seen that if the Lagrange multiplers λ^k happen to be optimal in (21), then the solution of the original nonlinear program (18) would also be optimal for this subproblem. Under favorable assumptions, and provided that the sequence $\{\mu_k\}$ is chosen judiciously, we find that the sequence (x^k, λ^k) converges to a point satisfying the optimality conditions (19).

Augmented Lagrangian methods were first proposed by Hestenes [11] and Powell [21]. Bertsekas' book [3] and the book of Conn, Gould, and Toint [7] describing the Lancelot code, were influential in later developments. The approach has proved particularly useful in "splitting" schemes, where the objective f is decomposed naturally into a sum of functions, each of which is assigned its own copy of the variable vector x. Equality of the different copies is enforced via equality constraints, and the augmented Lagrangian method is applied to the resulting equality-constrained problem. The appeal of this approach is that minimization with respect to each copy of x can be performed independently, and these individual minimizations may be simpler to perform than minimization of the original function f. Moreover, the possibility arises of performance these minimizations simultaneously, on a parallel computer.

7 Final Remarks

Our brief description of major problem classes in continuous optimization, and algorithms for solving them, has necessarily omitted several important topics. We mention several such topics before closing.

Stochastic and *robust* optimization deal with problems in which there is uncertainty in the objective functions or constraints, but where the uncertainty can be quantified and modeled. In these problems we may seek solutions that minimize the *expected value* of the uncertain objective, or that are guaranteed to satisfy the constraints with a certain specified probability.

Equilibrium problems are not optimization problems in that there is no objective to be minimized, but they use a range of algorithmic techniques that are closely related to optimization techniques. The basic formulation is as follows: Given a function $F : \Re^n \to \Re^n$, find a vector $x \in \Re^n$ such that

$$x \geq 0, \quad F(x) \geq 0, \quad x_i F_i(x) = 0, \; i = 1, 2, \ldots, n.$$

(Note that the KKT conditions in (7) have a similar form.) Equilibrium problems arose initially in economic applications and game theory. More recent classes of applications involve contact problems in mechanical simulations.

Nonlinear equations, in which we seek a vector $x \in \Re^n$ such that $F(x) = 0$, for some smooth function $F : \Re^n \to \Re^n$ arise throughout scientific computing. Newton's method, so fundamental in continuous optimization, is also key here. The Newton step is obtained by solving

$$\nabla F(x^k) d^k = -F(x^k),$$

(compare with (13)), where $\nabla F(x)$ is the $n \times n$ matrix of first partial derivatives of the components of $F(x)$.

Further Reading

1. Barzilai, J. and J. M. Borwein. 1988, Two-point step size gradient methods. *IMA Journal of Numerical Analysis* 8:141–148.
2. Beck, A. and M. Teboulle. 2009, A fast iterative shrinkage-threshold algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences* 2(1):183–202.
3. Bertsekas, D. P. 1982, *Constrained Optimization and Lagrange Multiplier Methods*. New York: Academic Press.
4. ———. 1999, *Nonlinear Programming*. Athena Scientific, second edition.
5. Boyd, S. and L. Vandenberghe. 2003, *Convex Optimization*. Cambridge University Press.
6. Clarke, F. H. 1983, *Optimization and Nonsmooth Analysis*. New York: John Wiley.
7. Conn, A. R., N. I. M. Gould, and P. Toint. 1992, *LANCELOT: A Fortran package for large-scale nonlinear optimization*, *Springer Series in Computational Mathematics*, volume 17. Springer-Verlag.
8. Conn, A. R., K. Scheinberg, and L. N. Vicente. 2009, *Introduction to Derivative-Free Optimization*, *MPS-SIAM Series in Optimization*, volume 8. SIAM.
9. Dantzig, G. B. 1963, *Linear Programming and Extensions*. Princeton, New Jersey: Princeton University Press.
10. Griewank, A. and A. Walther. 2008, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia, PA: SIAM, second edition.
11. Hestenes, M. R. 1969, Multiplier and gradient methods. *Journal of Optimization Theory and Applications* 4:303–320.

12. Karmarkar, N. 1984, A new polynomial-time algorithm for linear programming. *Combinatorica* 4:373–395.
13. Khachiyan, L. G. 1979, A polynomial algorithm in linear programming. *Soviet Mathematics Doklady* 20:191–194.
14. Klee, V. and G. J. Minty. 1972, How good is the simplex algorithm? In *Inequalities*, edited by O. Shisha, pp. 159–175, New York: Academic Press.
15. Liu, D. C. and J. Nocedal. 1989, On the limited-memory BFGS method for large scale optimization. *Mathematical Programming* 45:503–528.
16. Nelder, J. A. and R. Mead. 1965, A simplex method for function minimization. *Computer Journal* 7:308–313.
17. Nemirovski, A., A. Juditsky, G. Lan, and A. Shapiro. 2009, Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization* 19(4):1574–1609.
18. Nesterov, Y. 2004, *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers.
19. Nesterov, Y. and A. S. Nemirovskii. 1994, *Interior Point Polynomial Methods in Convex Programming*. Philadelphia: SIAM Publications.
20. Nocedal, J. and S. J. Wright. 2006, *Numerical Optimization*. New York: Springer, second edition.
21. Powell, M. J. D. 1969, A method for nonlinear constraints in minimization problems. In *Optimization*, edited by R. Fletcher, pp. 283–298, New York: Academic Press.
22. Rockafellar, R. T. 1970, *Convex Analysis*. Princeton, N.J.: Princeton University Press.
23. Rockafellar, R. T. and R. J. Wets. 1998, *Variational Analysis*. Berlin: Springer.
24. Spielman, D. A. and S.-H. Teng. 2004, Smoothed analysis of algorithms: Why the simplex method usually takes polynomial time. *Journal of the Association for Computing Machinery* 51(3):385–463.
25. Todd, M. J. 2001, Semidefinite optimization. *Acta Numerica* 10:515–560.
26. Wächter, A. and L. T. Biegler. 2006, On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming, Series B* 106(1):25–57.
27. Wright, S. J. 1997, *Primal-Dual Interior-Point Methods*. Philadelphia, PA: SIAM.

Biography

Stephen Wright is a Professor of Computer Sciences at the University of Wisconsin-Madison.