# Chapter 1

# Introduction

In this introduction we give an overview of mathematical optimization, focusing on the special role of convex optimization. The concepts introduced informally here will be covered in later chapters, with more care and technical detail.

## 1.1 Mathematical optimization

A *mathematical optimization problem*, or just *optimization problem*, has the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, m. \end{array} \tag{1.1}$$

Here the vector $x = (x_1, \ldots, x_n)$ is the *optimization variable* of the problem, the function $f_0 : \mathbf{R}^n \to \mathbf{R}$ is the *objective* function, the functions $f_i : \mathbf{R}^n \to \mathbf{R}$, $i = 1, \ldots, m$, are the (inequality) *constraint functions*, and the constants $b_1, \ldots, b_m$ are the limits, or bounds, for the constraints. A vector $x^\star$ is called *optimal*, or a *solution* of the problem (1.1), if it has the smallest objective value among all vectors that satisfy the constraints: for any $z$ with $f_1(z) \leq b_1, \ldots, f_m(z) \leq b_m$, we have $f_0(z) \geq f_0(x^\star)$.

We generally consider families or classes of optimization problems, characterized by particular forms of the objective and constraint functions. As an important example, the optimization problem (1.1) is called a *linear program* if the objective and constraint functions $f_0, \ldots, f_m$ are linear, *i.e.*, satisfy

$$f_i(\alpha x + \beta y) = \alpha f_i(x) + \beta f_i(y) \tag{1.2}$$

for all $x$, $y \in \mathbf{R}^n$ and all $\alpha$, $\beta \in \mathbf{R}$. If the optimization problem is not linear, it is called a *nonlinear program*.

This book is about a class of optimization problems called *convex optimization problems*. A convex optimization problem is one in which the objective and constraint functions are convex, which means they satisfy the inequality

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y) \tag{1.3}$$

for all $x$, $y \in \mathbf{R}^n$ and all $\alpha$, $\beta \in \mathbf{R}$ with $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$. Comparing (1.3) and (1.2), we see that convexity is more general than linearity: inequality replaces the more restrictive equality, and the inequality must hold only for certain values of $\alpha$ and $\beta$. Since any linear program is therefore a convex optimization problem, we can consider convex optimization to be a generalization of linear programming.

### 1.1.1 Applications

The optimization problem (1.1) is an abstraction of the problem of making the best possible choice of a vector in $\mathbf{R}^n$ from a set of candidate choices. The variable $x$ represents the choice made; the constraints $f_i(x) \leq b_i$ represent firm requirements or specifications that limit the possible choices, and the objective value $f_0(x)$ represents the cost of choosing $x$. (We can also think of $-f_0(x)$ as representing the value, or utility, of choosing $x$.) A solution of the optimization problem (1.1) corresponds to a choice that has minimum cost (or maximum utility), among all choices that meet the firm requirements.

In *portfolio optimization*, for example, we seek the best way to invest some capital in a set of $n$ assets. The variable $x_i$ represents the investment in the $i$th asset, so the vector $x \in \mathbf{R}^n$ describes the overall portfolio allocation across the set of assets. The constraints might represent a limit on the budget (*i.e.*, a limit on the total amount to be invested), the requirement that investments are nonnegative (assuming short positions are not allowed), and a minimum acceptable value of expected return for the whole portfolio. The objective or cost function might be a measure of the overall risk or variance of the portfolio return. In this case, the optimization problem (1.1) corresponds to choosing a portfolio allocation that minimizes risk, among all possible allocations that meet the firm requirements.

Another example is *device sizing* in electronic design, which is the task of choosing the width and length of each device in an electronic circuit. Here the variables represent the widths and lengths of the devices. The constraints represent a variety of engineering requirements, such as limits on the device sizes imposed by the manufacturing process, timing requirements that ensure that the circuit can operate reliably at a specified speed, and a limit on the total area of the circuit. A common objective in a device sizing problem is the total power consumed by the circuit. The optimization problem (1.1) is to find the device sizes that satisfy the design requirements (on manufacturability, timing, and area) and are most power efficient. 1.1.2

In *data fitting*, the task is to find a model, from a family of potential models, that best fits some observed data and prior information. Here the variables are the parameters in the model, and the constraints can represent prior information or required limits on the parameters (such as nonnegativity). The objective function might be a measure of misfit or prediction error between the observed data and the values predicted by the model, or a statistical measure of the unlikeliness or implausibility of the parameter values. The optimization problem (1.1) is to find the model parameter values that are consistent with the prior information, and give the smallest misfit or prediction error with the observed data (or, in a statistical

framework, are most likely).

An amazing variety of practical problems involving decision making (or system design, analysis, and operation) can be cast in the form of a mathematical optimization problem, or some variation such as a multicriterion optimization problem. Indeed, mathematical optimization has become an important tool in many areas. It is widely used in engineering, in electronic design automation, automatic control systems, and optimal design problems arising in civil, chemical, mechanical, and aerospace engineering. Optimization is used for problems arising in network design and operation, finance, supply chain management, scheduling, and many other areas. The list of applications is still steadily expanding.

For most of these applications, mathematical optimization is used as an aid to a human decision maker, system designer, or system operator, who supervises the process, checks the results, and modifies the problem (or the solution approach) when necessary. This human decision maker also carries out any actions suggested by the optimization problem, *e.g.*, buying or selling assets to achieve the optimal portfolio.

A relatively recent phenomenon opens the possibility of many other applications for mathematical optimization. With the proliferation of computers embedded in products, we have seen a rapid growth in *embedded optimization*. In these embedded applications, optimization is used to automatically make real-time choices, and even carry out the associated actions, with no (or little) human intervention or oversight. In some application areas, this blending of traditional automatic control systems and embedded optimization is well under way; in others, it is just starting. Embedded real-time optimization raises some new challenges: in particular, it requires solution methods that are extremely reliable, and solve problems in a predictable amount of time (and memory).

### 1.1.2 Solving optimization problems

A *solution method* for a class of optimization problems is an algorithm that computes a solution of the problem (to some given accuracy), given a particular problem from the class, *i.e.*, an *instance* of the problem. Since the late 1940s, a large effort has gone into developing algorithms for solving various classes of optimization problems, analyzing their properties, and developing good software implementations. The effectiveness of these algorithms, *i.e.*, our ability to solve the optimization problem (1.1), varies considerably, and depends on factors such as the particular forms of the objective and constraint functions, how many variables and constraints there are, and special structure, such as *sparsity*. (A problem is *sparse* if each constraint function depends on only a small number of the variables).

Even when the objective and constraint functions are smooth (for example, polynomials) the general optimization problem (1.1) is surprisingly difficult to solve. Approaches to the general problem therefore involve some kind of compromise, such as very long computation time, or the possibility of not finding the solution. Some of these methods are discussed in §1.4.

There are, however, some important exceptions to the general rule that most optimization problems are difficult to solve. For a few problem classes we have

effective algorithms that can reliably solve even large problems, with hundreds or thousands of variables and constraints. Two important and well known examples, described in §1.2 below (and in detail in chapter 4), are least-squares problems and linear programs. It is less well known that convex optimization is another exception to the rule: Like least-squares or linear programming, there are very effective algorithms that can reliably and efficiently solve even large convex problems.

## 1.2 Least-squares and linear programming

In this section we describe two very widely known and used special subclasses of convex optimization: least-squares and linear programming. (A complete technical treatment of these problems will be given in chapter 4.)

### 1.2.1 Least-squares problems

A *least-squares* problem is an optimization problem with no constraints (*i.e.*, $m = 0$) and an objective which is a sum of squares of terms of the form $a_i^T x - b_i$:

$$\text{minimize} \quad f_0(x) = \|Ax - b\|_2^2 = \sum_{i=1}^{k} (a_i^T x - b_i)^2. \tag{1.4}$$

Here $A \in \mathbf{R}^{k \times n}$ (with $k \geq n$), $a_i^T$ are the rows of $A$, and the vector $x \in \mathbf{R}^n$ is the optimization variable.

#### Solving least-squares problems

The solution of a least-squares problem (1.4) can be reduced to solving a set of linear equations,

$$(A^T A)x = A^T b,$$

so we have the analytical solution $x = (A^T A)^{-1} A^T b$. For least-squares problems we have good algorithms (and software implementations) for solving the problem to high accuracy, with very high reliability. The least-squares problem can be solved in a time approximately proportional to $n^2 k$, with a known constant. A current desktop computer can solve a least-squares problem with hundreds of variables, and thousands of terms, in a few seconds; more powerful computers, of course, can solve larger problems, or the same size problems, faster. (Moreover, these solution times will decrease exponentially in the future, according to Moore's law.) Algorithms and software for solving least-squares problems are reliable enough for embedded optimization.

In many cases we can solve even larger least-squares problems, by exploiting some special structure in the coefficient matrix $A$. Suppose, for example, that the matrix $A$ is *sparse*, which means that it has far fewer than $kn$ nonzero entries. By exploiting sparsity, we can usually solve the least-squares problem much faster than order $n^2 k$. A current desktop computer can solve a sparse least-squares problem

with tens of thousands of variables, and hundreds of thousands of terms, in around a minute (although this depends on the particular sparsity pattern).

For extremely large problems (say, with millions of variables), or for problems with exacting real-time computing requirements, solving a least-squares problem can be a challenge. But in the vast majority of cases, we can say that existing methods are very effective, and extremely reliable. Indeed, we can say that solving least-squares problems (that are not on the boundary of what is currently achievable) is a (mature) *technology*, that can be reliably used by many people who do not know, and do not need to know, the details.

### Using least-squares

The least-squares problem is the basis for regression analysis, optimal control, and many parameter estimation and data fitting methods. It has a number of statistical interpretations, *e.g.*, as maximum likelihood estimation of a vector $x$, given linear measurements corrupted by Gaussian measurement errors.

Recognizing an optimization problem as a least-squares problem is straightforward; we only need to verify that the objective is a quadratic function (and then test whether the associated quadratic form is positive semidefinite). While the basic least-squares problem has a simple fixed form, several standard techniques are used to increase its flexibility in applications.

In *weighted least-squares*, the weighted least-squares cost

$$\sum_{i=1}^{k} w_i(a_i^T x - b_i)^2,$$

where $w_1, \ldots, w_k$ are positive, is minimized. (This problem is readily cast and solved as a standard least-squares problem.) Here the weights $w_i$ are chosen to reflect differing levels of concern about the sizes of the terms $a_i^T x - b_i$, or simply to influence the solution. In a statistical setting, weighted least-squares arises in estimation of a vector $x$, given linear measurements corrupted by errors with unequal variances.

Another technique in least-squares is *regularization*, in which extra terms are added to the cost function. In the simplest case, a positive multiple of the sum of squares of the variables is added to the cost function:

$$\sum_{i=1}^{k}(a_i^T x - b_i)^2 + \rho \sum_{i=1}^{n} x_i^2,$$

where $\rho > 0$. (This problem too can be formulated as a standard least-squares problem.) The extra terms penalize large values of $x$, and result in a sensible solution in cases when minimizing the first sum only does not. The parameter $\rho$ is chosen by the user to give the right trade-off between making the original objective function $\sum_{i=1}^{k}(a_i^T x - b_i)^2$ small, while keeping $\sum_{i=1}^{n} x_i^2$ not too big. Regularization comes up in statistical estimation when the vector $x$ to be estimated is given a prior distribution.

Weighted least-squares and regularization are covered in chapter 6; their statistical interpretations are given in chapter 7.

### 1.2.2 Linear programming

Another important class of optimization problems is *linear programming*, in which the objective and all constraint functions are linear:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & a_i^T x \leq b_i, \quad i = 1, \ldots, m. \end{array} \tag{1.5}$$

Here the vectors $c, a_1, \ldots, a_m \in \mathbf{R}^n$ and scalars $b_1, \ldots, b_m \in \mathbf{R}$ are problem parameters that specify the objective and constraint functions.

#### Solving linear programs

There is no simple analytical formula for the solution of a linear program (as there is for a least-squares problem), but there are a variety of very effective methods for solving them, including Dantzig's simplex method, and the more recent interior-point methods described later in this book. While we cannot give the exact number of arithmetic operations required to solve a linear program (as we can for least-squares), we can establish rigorous bounds on the number of operations required to solve a linear program, to a given accuracy, using an interior-point method. The complexity in practice is order $n^2 m$ (assuming $m \geq n$) but with a constant that is less well characterized than for least-squares. These algorithms are quite reliable, although perhaps not quite as reliable as methods for least-squares. We can easily solve problems with hundreds of variables and thousands of constraints on a small desktop computer, in a matter of seconds. If the problem is sparse, or has some other exploitable structure, we can often solve problems with tens or hundreds of thousands of variables and constraints.

As with least-squares problems, it is still a challenge to solve extremely large linear programs, or to solve linear programs with exacting real-time computing requirements. But, like least-squares, we can say that solving (most) linear programs is a mature technology. Linear programming solvers can be (and are) embedded in many tools and applications.

#### Using linear programming

Some applications lead directly to linear programs in the form (1.5), or one of several other standard forms. In many other cases the original optimization problem does not have a standard linear program form, but can be transformed to an equivalent linear program (and then, of course, solved) using techniques covered in detail in chapter 4.

As a simple example, consider the *Chebyshev approximation problem*:

$$\text{minimize} \quad \max_{i=1,\ldots,k} |a_i^T x - b_i|. \tag{1.6}$$

Here $x \in \mathbf{R}^n$ is the variable, and $a_1, \ldots, a_k \in \mathbf{R}^n$, $b_1, \ldots, b_k \in \mathbf{R}$ are parameters that specify the problem instance. Note the resemblance to the least-squares problem (1.4). For both problems, the objective is a measure of the size of the terms $a_i^T x - b_i$. In least-squares, we use the sum of squares of the terms as objective, whereas in Chebyshev approximation, we use the maximum of the absolute values.

One other important distinction is that the objective function in the Chebyshev approximation problem (1.6) is not differentiable; the objective in the least-squares problem (1.4) is quadratic, and therefore differentiable.

The Chebyshev approximation problem (1.6) can be solved by solving the linear program

$$\begin{array}{ll} \text{minimize} & t \\ \text{subject to} & a_i^T x - t \leq b_i, \quad i = 1, \ldots, k \\ & -a_i^T x - t \leq -b_i, \quad i = 1, \ldots, k, \end{array} \tag{1.7}$$

with variables $x \in \mathbf{R}^n$ and $t \in \mathbf{R}$. (The details will be given in chapter 4.) Since linear programs are readily solved, the Chebyshev approximation problem is therefore readily solved.

Anyone with a working knowledge of linear programming would recognize the Chebyshev approximation problem (1.6) as one that can be reduced to a linear program. For those without this background, though, it might not be obvious that the Chebyshev approximation problem (1.6), with its nondifferentiable objective, can be formulated and solved as a linear program.

While recognizing problems that can be reduced to linear programs is more involved than recognizing a least-squares problem, it is a skill that is readily acquired, since only a few standard tricks are used. The task can even be partially automated; some software systems for specifying and solving optimization problems can automatically recognize (some) problems that can be reformulated as linear programs.

## 1.3 Convex optimization

A convex optimization problem is one of the form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, m, \end{array} \tag{1.8}$$

where the functions $f_0, \ldots, f_m : \mathbf{R}^n \to \mathbf{R}$ are convex, *i.e.*, satisfy

$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$$

for all $x$, $y \in \mathbf{R}^n$ and all $\alpha$, $\beta \in \mathbf{R}$ with $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$. The least-squares problem (1.4) and linear programming problem (1.5) are both special cases of the general convex optimization problem (1.8).

### 1.3.1 Solving convex optimization problems

There is in general no analytical formula for the solution of convex optimization problems, but (as with linear programming problems) there are very effective methods for solving them. Interior-point methods work very well in practice, and in some cases can be proved to solve the problem to a specified accuracy with a number of

operations that does not exceed a polynomial of the problem dimensions. (This is covered in chapter 11.)

We will see that interior-point methods can solve the problem (1.8) in a number of steps or iterations that is almost always in the range between 10 and 100. Ignoring any structure in the problem (such as sparsity), each step requires on the order of

$$\max\{n^3, n^2m, F\}$$

operations, where $F$ is the cost of evaluating the first and second derivatives of the objective and constraint functions $f_0, \ldots, f_m$.

Like methods for solving linear programs, these interior-point methods are quite reliable. We can easily solve problems with hundreds of variables and thousands of constraints on a current desktop computer, in at most a few tens of seconds. By exploiting problem structure (such as sparsity), we can solve far larger problems, with many thousands of variables and constraints.

We cannot yet claim that solving general convex optimization problems is a mature technology, like solving least-squares or linear programming problems. Research on interior-point methods for general nonlinear convex optimization is still a very active research area, and no consensus has emerged yet as to what the best method or methods are. But it is reasonable to expect that solving general convex optimization problems will become a technology within a few years. And for some subclasses of convex optimization problems, for example second-order cone programming or geometric programming (studied in detail in chapter 4), it is fair to say that interior-point methods are approaching a technology.

### 1.3.2 Using convex optimization

Using convex optimization is, at least conceptually, very much like using least-squares or linear programming. If we can formulate a problem as a convex optimization problem, then we can solve it efficiently, just as we can solve a least-squares problem efficiently. With only a bit of exaggeration, we can say that, if you formulate a practical problem as a convex optimization problem, then you have solved the original problem.

There are also some important differences. Recognizing a least-squares problem is straightforward, but recognizing a convex function can be difficult. In addition, there are many more tricks for transforming convex problems than for transforming linear programs. Recognizing convex optimization problems, or those that can be transformed to convex optimization problems, can therefore be challenging. The main goal of this book is to give the reader the background needed to do this. Once the skill of recognizing or formulating convex optimization problems is developed, you will find that surprisingly many problems can be solved via convex optimization.

The challenge, and art, in using convex optimization is in recognizing and formulating the problem. Once this formulation is done, solving the problem is, like least-squares or linear programming, (almost) technology.

## 1.4 Nonlinear optimization

*Nonlinear optimization* (or nonlinear programming) is the term used to describe an optimization problem when the objective or constraint functions are not linear, but not known to be convex. Sadly, there are no effective methods for solving the general nonlinear programming problem (1.1). Even simple looking problems with as few as ten variables can be extremely challenging, while problems with a few hundreds of variables can be intractable. Methods for the general nonlinear programming problem therefore take several different approaches, each of which involves some compromise.

### 1.4.1 Local optimization

In *local optimization*, the compromise is to give up seeking the optimal $x$, which minimizes the objective over all feasible points. Instead we seek a point that is only locally optimal, which means that it minimizes the objective function among feasible points that are near it, but is not guaranteed to have a lower objective value than all other feasible points. A large fraction of the research on general nonlinear programming has focused on methods for local optimization, which as a consequence are well developed.

Local optimization methods can be fast, can handle large-scale problems, and are widely applicable, since they only require differentiability of the objective and constraint functions. As a result, local optimization methods are widely used in applications where there is value in finding a good point, if not the very best. In an engineering design application, for example, local optimization can be used to improve the performance of a design originally obtained by manual, or other, design methods.

There are several disadvantages of local optimization methods, beyond (possibly) not finding the true, globally optimal solution. The methods require an initial guess for the optimization variable. This initial guess or starting point is critical, and can greatly affect the objective value of the local solution obtained. Little information is provided about how far from (globally) optimal the local solution is. Local optimization methods are often sensitive to algorithm parameter values, which may need to be adjusted for a particular problem, or family of problems.

Using a local optimization method is trickier than solving a least-squares problem, linear program, or convex optimization problem. It involves experimenting with the choice of algorithm, adjusting algorithm parameters, and finding a good enough initial guess (when one instance is to be solved) or a method for producing a good enough initial guess (when a family of problems is to be solved). Roughly speaking, local optimization methods are more art than technology. Local optimization is a well developed art, and often very effective, but it is nevertheless an art. In contrast, there is little art involved in solving a least-squares problem or a linear program (except, of course, those on the boundary of what is currently possible).

An interesting comparison can be made between local optimization methods for nonlinear programming, and convex optimization. Since differentiability of the ob-

jective and constraint functions is the only requirement for most local optimization methods, formulating a practical problem as a nonlinear optimization problem is relatively straightforward. The art in local optimization is in solving the problem (in the weakened sense of finding a locally optimal point), once it is formulated. In convex optimization these are reversed: The art and challenge is in problem formulation; once a problem is formulated as a convex optimization problem, it is relatively straightforward to solve it.

### 1.4.2 Global optimization

In *global optimization*, the true global solution of the optimization problem (1.1) is found: the compromise is efficiency. The worst-case complexity of global optimization methods grows exponentially with the problem sizes $n$ and $m$; the hope is that in practice, for the particular problem instances encountered, the method is far faster. While this favorable situation does occur, it is not typical. Even small problems, with a few tens of variables, can take a very long time (*e.g.*, hours or days) to solve.

Global optimization is used for problems with a small number of variables, where computing time is not critical, and the value of finding the true global solution is very high. One example from engineering design is *worst-case analysis* or *verification* of a high value or safety-critical system. Here the variables represent uncertain parameters, that can vary during manufacturing, or with the environment or operating condition. The objective function is a utility function, *i.e.*, one for which smaller values are worse than larger values, and the constraints represent prior knowledge about the possible parameter values. The optimization problem (1.1) is the problem of finding the *worst-case* values of the parameters. If the worst-case value is acceptable, we can certify the system as safe or reliable (with respect to the parameter variations).

A local optimization method can rapidly find a set of parameter values that is bad, but not guaranteed to be the absolute worst possible. If a local optimization method finds parameter values that yield unacceptable performance, it has succeeded in determining that the system is not reliable. But a local optimization method cannot certify the system as reliable; it can only fail to find bad parameter values. A global optimization method, in contrast, will find the absolute worst values of the parameters, and if the associated performance is acceptable, can certify the system as safe. The cost is computation time, which can be very large, even for a relatively small number of parameters. But it may be worth it in cases where the value of certifying the performance is high, or the cost of being wrong about the reliability or safety is high.

### 1.4.3 Role of convex optimization in nonconvex problems

In this book we focus primarily on convex optimization problems, and applications that can be reduced to convex optimization problems. But convex optimization also plays an important role in problems that are *not* convex.

#### Initialization for local optimization

One obvious use is to combine convex optimization with a local optimization method. Starting with a nonconvex problem, we first find an approximate, but convex, formulation of the problem. By solving this approximate problem, which can be done easily and without an initial guess, we obtain the exact solution to the approximate convex problem. This point is then used as the starting point for a local optimization method, applied to the original nonconvex problem.

#### Convex heuristics for nonconvex optimization

Convex optimization is the basis for several heuristics for solving nonconvex problems. One interesting example we will see is the problem of finding a *sparse* vector $x$ (*i.e.*, one with few nonzero entries) that satisfies some constraints. While this is a difficult combinatorial problem, there are some simple heuristics, based on convex optimization, that often find fairly sparse solutions. (These are described in chapter 6.)

Another broad example is given by *randomized algorithms*, in which an approximate solution to a nonconvex problem is found by drawing some number of candidates from a probability distribution, and taking the best one found as the approximate solution. Now suppose the family of distributions from which we will draw the candidates is parametrized, *e.g.*, by its mean and covariance. We can then pose the question, which of these distributions gives us the smallest expected value of the objective? It turns out that this problem is sometimes a convex problem, and therefore efficiently solved. (See, *e.g.*, exercise 11.23.)

#### Bounds for global optimization

Many methods for global optimization require a cheaply computable lower bound on the optimal value of the nonconvex problem. Two standard methods for doing this are based on convex optimization. In *relaxation*, each nonconvex constraint is replaced with a looser, but convex, constraint. In *Lagrangian relaxation*, the Lagrangian dual problem (described in chapter 5) is solved. This problem is convex, and provides a lower bound on the optimal value of the nonconvex problem.

## 1.5 Outline

The book is divided into three main parts, titled *Theory*, *Applications*, and *Algorithms*.

### 1.5.1 Part I: Theory

In part I, *Theory*, we cover basic definitions, concepts, and results from convex analysis and convex optimization. We make no attempt to be encyclopedic, and skew our selection of topics toward those that we think are useful in recognizing