B.Sc(H) Computer Science 3rd Sem
Numerical Optimization
Guidelines

Unit	Content with chapters & page nos.	Reference	Duration
1	Chapter1+ Solution by graphical method	[1]	13 Hrs
	Simplex Method with all special cases chapter 5 till 5.6	[3]	
2	2.1 without proof of theorems	[1]	12 Hrs
	9.1-9.6	[2]	
3	9.7 till pg 350	[2]	6 Hrs
4	8.1	[1]	6 Hrs
5	12.1 till pg 313 + Langrangian Method Numerical Approach	[1]	8 Hrs

Note: Proof of all theorems and Lemmas can be skipped

Essential/recommended readings

1. J. Nocedal and S.J. Wright, *Numerical Optimization*, 2nd edition, Springer Series inOperations Research, 2006.
2. A, Mehra, S Chandra, Jayadeva, *Numerical Optimization with Applications*, NarosaPublishing House, New Delhi, 2009,
3. J. Matousek and Bernd Gartner, *Understanding and using Linear programming*, Springer

Practicals must be done in Python

Practical list

1. WAP for finding optimal solution using Line Search method.
2. WAP to solve a LPP graphically.
3. WAP to compute the gradient and Hessian of the function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

4. WAP to find Global Optimal Solution of a function
   $f(x) = -10Cos(\pi x - 2.2) + (x + 1.5)x$ algebraically
5. WAP to find Global Optimal Solution of a function
   $f(x) = -10Cos(\pi x - 2.2) + (x + 1.5)x$ graphically
6. WAP to solve constraint optimization problem.

Chapter 1

Introduction

People optimize. Investors seek to create portfolios that avoid excessive risk while achieving a high rate of return. Manufacturers aim for maximum efficiency in the design and operation of their production processes. Engineers adjust parameters to optimize the performance of their designs.

Nature optimizes. Physical systems tend to a state of minimum energy. The molecules in an isolated chemical system react with each other until the total potential energy of their electrons is minimized. Rays of light follow paths that minimize their travel time.

Optimization is an important tool in decision science and in the analysis of physical systems. To make use of this tool, we must first identify some *objective*, a quantitative measure of the performance of the system under study. This objective could be profit, time, potential energy, or any quantity or combination of quantities that can be represented by a single number. The objective depends on certain characteristics of the system, called *variables* or *unknowns*. Our goal is to find values of the variables that optimize the objective. Often the variables are restricted, or *constrained*, in some way. For instance, quantities such as electron density in a molecule and the interest rate on a loan cannot be negative.

The process of identifying objective, variables, and constraints for a given problem is known as *modeling*. Construction of an appropriate model is the first step—sometimes the most important step—in the optimization process. If the model is too simplistic, it will not give useful insights into the practical problem. If it is too complex, it may be too difficult to solve.

Once the model has been formulated, an optimization algorithm can be used to find its solution, usually with the help of a computer. There is no universal optimization algorithm but rather a collection of algorithms, each of which is tailored to a particular type of optimization problem. The responsibility of choosing the algorithm that is appropriate for a specific application often falls on the user. This choice is an important one, as it may determine whether the problem is solved rapidly or slowly and, indeed, whether the solution is found at all.

After an optimization algorithm has been applied to the model, we must be able to recognize whether it has succeeded in its task of finding a solution. In many cases, there are elegant mathematical expressions known as *optimality conditions* for checking that the current set of variables is indeed the solution of the problem. If the optimality conditions are not satisfied, they may give useful information on how the current estimate of the solution can be improved. The model may be improved by applying techniques such as *sensitivity analysis*, which reveals the sensitivity of the solution to changes in the model and data. Interpretation of the solution in terms of the application may also suggest ways in which the model can be refined or improved (or corrected). If any changes are made to the model, the optimization problem is solved anew, and the process repeats.

MATHEMATICAL FORMULATION

Mathematically speaking, optimization is the minimization or maximization of a function subject to constraints on its variables. We use the following notation:

- x is the vector of *variables*, also called *unknowns* or *parameters*;
- f is the *objective function*, a (scalar) function of x that we want to maximize or minimize;
- c_i are *constraint* functions, which are scalar functions of x that define certain equations and inequalities that the unknown vector x must satisfy.

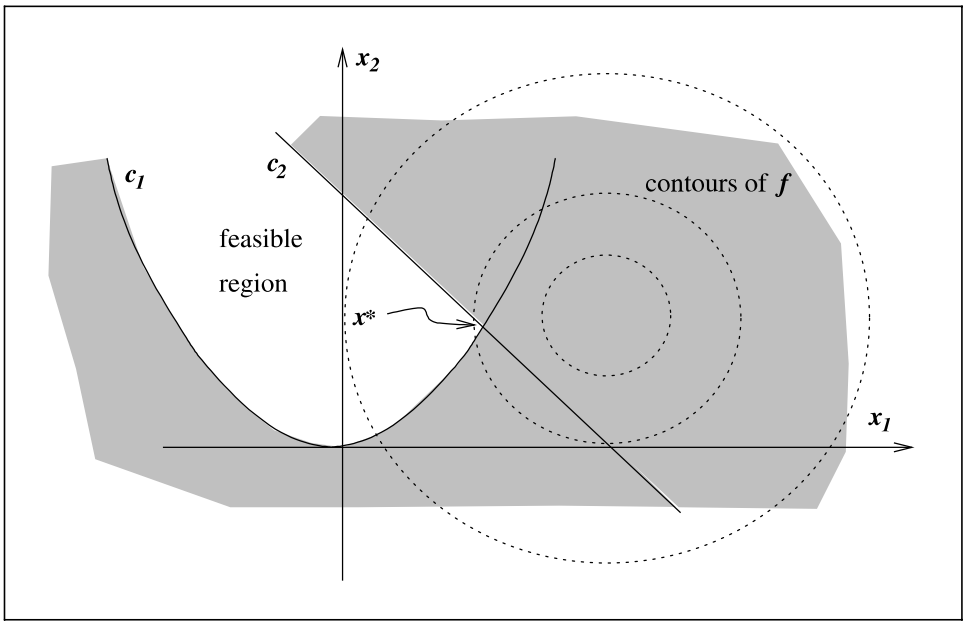


Figure 1.1 Geometrical representation of the problem (1.2).

Using this notation, the optimization problem can be written as follows:

$$\min_{x \in \mathbb{R}^n} f(x) \qquad \text{subject to} \quad \begin{array}{ll} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}. \end{array} \tag{1.1}$$

Here $\mathcal{I}$ and $\mathcal{E}$ are sets of indices for equality and inequality constraints, respectively.

As a simple example, consider the problem

$$\min\ (x_1 - 2)^2 + (x_2 - 1)^2 \qquad \text{subject to} \quad \begin{array}{l} x_1^2 - x_2 \leq 0, \\ x_1 + x_2 \leq 2. \end{array} \tag{1.2}$$

We can write this problem in the form (1.1) by defining

$$f(x) = (x_1 - 2)^2 + (x_2 - 1)^2, \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

$$c(x) = \begin{bmatrix} c_1(x) \\ c_2(x) \end{bmatrix} = \begin{bmatrix} -x_1^2 + x_2 \\ -x_1 - x_2 + 2 \end{bmatrix}, \qquad \mathcal{I} = \{1, 2\}, \quad \mathcal{E} = \emptyset.$$

Figure 1.1 shows the contours of the objective function, that is, the set of points for which $f(x)$ has a constant value. It also illustrates the *feasible region*, which is the set of points satisfying all the constraints (the area between the two constraint boundaries), and the point

x^*, which is the solution of the problem. Note that the "infeasible side" of the inequality constraints is shaded.

The example above illustrates, too, that transformations are often necessary to express an optimization problem in the particular form (1.1). Often it is more natural or convenient to label the unknowns with two or three subscripts, or to refer to different variables by completely different names, so that relabeling is necessary to pose the problem in the form (1.1). Another common difference is that we are required to *maximize* rather than minimize f, but we can accommodate this change easily by *minimizing* $-f$ in the formulation (1.1). Good modeling systems perform the conversion to standardized formulations such as (1.1) transparently to the user.

EXAMPLE: A TRANSPORTATION PROBLEM

We begin with a much simplified example of a problem that might arise in manufacturing and transportation. A chemical company has 2 factories F_1 and F_2 and a dozen retail outlets $R_1, R_2, \ldots, R_{12}$. Each factory F_i can produce a_i tons of a certain chemical product each week; a_i is called the *capacity* of the plant. Each retail outlet R_j has a known weekly *demand* of b_j tons of the product. The cost of shipping one ton of the product from factory F_i to retail outlet R_j is c_{ij}.

The problem is to determine how much of the product to ship from each factory to each outlet so as to satisfy all the requirements and minimize cost. The variables of the problem are x_{ij}, $i = 1, 2$, $j = 1, \ldots, 12$, where x_{ij} is the number of tons of the product shipped from factory F_i to retail outlet R_j; see Figure 1.2. We can write the problem as

$$\min \sum_{ij} c_{ij} x_{ij} \tag{1.3a}$$

$$\text{subject to} \sum_{j=1}^{12} x_{ij} \leq a_i, \quad i = 1, 2, \tag{1.3b}$$

$$\sum_{i=1}^{2} x_{ij} \geq b_j, \quad j = 1, \ldots, 12, \tag{1.3c}$$

$$x_{ij} \geq 0, \quad i = 1, 2, \quad j = 1, \ldots, 12. \tag{1.3d}$$

This type of problem is known as a *linear programming* problem, since the objective function and the constraints are all linear functions. In a more practical model, we would also include costs associated with manufacturing and storing the product. There may be volume discounts in practice for shipping the product; for example the cost (1.3a) could be represented by $\sum_{ij} c_{ij}\sqrt{\delta + x_{ij}}$, where $\delta > 0$ is a small subscription fee. In this case, the problem is a *nonlinear program* because the objective function is nonlinear.

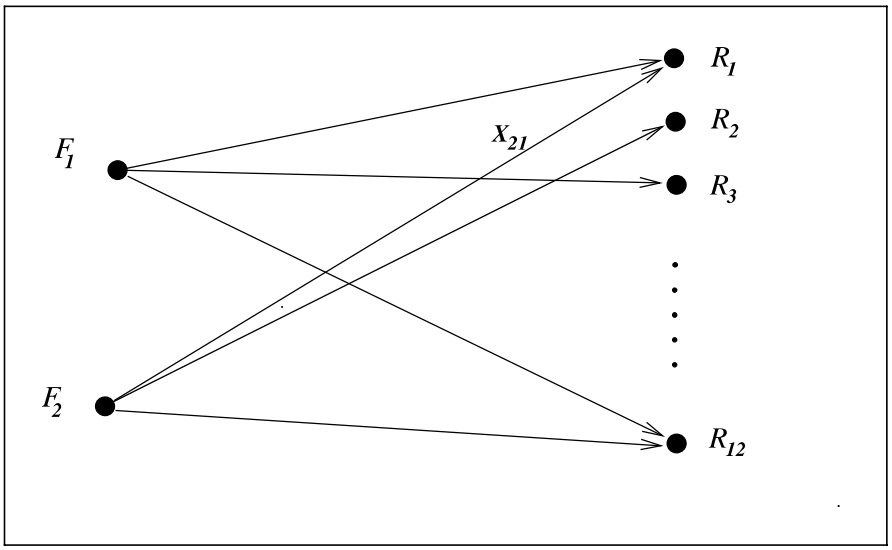


Figure 1.2 A transportation problem.

CONTINUOUS VERSUS DISCRETE OPTIMIZATION

In some optimization problems the variables make sense only if they take on integer values. For example, a variable x_i could represent the number of power plants of type i that should be constructed by an electicity provider during the next 5 years, or it could indicate whether or not a particular factory should be located in a particular city. The mathematical formulation of such problems includes integrality constraints, which have the form $x_i \in \mathbf{Z}$, where $\mathbf{Z}$ is the set of integers, or binary constraints, which have the form $x_i \in \{0, 1\}$, in addition to algebraic constraints like those appearing in (1.1). Problems of this type are called *integer programming* problems. If some of the variables in the problem are *not* restricted to be integer or binary variables, they are sometimes called *mixed integer programming* problems, or MIPs for short.

Integer programming problems are a type of *discrete optimization* problem. Generally, discrete optimization problems may contain not only integers and binary variables, but also more abstract variable objects such as permutations of an ordered set. The defining feature of a discrete optimization problem is that the unknown x is drawn from a a finite (but often very large) set. By contrast, the feasible set for *continuous optimization* problems—the class of problems studied in this book—is usually uncountably infinite, as when the components of x are allowed to be real numbers. Continuous optimization problems are normally easier to solve because the smoothness of the functions makes it possible to use objective and constraint information at a particular point x to deduce information about the function's behavior at all points close to x. In discrete problems, by constrast, the behavior of the objective and constraints may change significantly as we move from one feasible point to another, even if the two points are "close" by some measure. The feasible sets for discrete optimization problems can be thought of as exhibiting an extreme form of nonconvexity, as a convex combination of two feasible points is in general not feasible.

Discrete optimization problems are not addressed directly in this book; we refer the reader to the texts by Papadimitriou and Steiglitz [235], Nemhauser and Wolsey [224], Cook et al. [77], and Wolsey [312] for comprehensive treatments of this subject. We note, however, that continuous optimization techniques often play an important role in solving discrete optimization problems. For instance, the branch-and-bound method for integer linear programming problems requires the repeated solution of linear programming "relaxations," in which some of the integer variables are fixed at integer values, while for other integer variables the integrality constraints are temporarily ignored. These subproblems are usually solved by the simplex method, which is discussed in Chapter 13 of this book.

CONSTRAINED AND UNCONSTRAINED OPTIMIZATION

Problems with the general form (1.1) can be classified according to the nature of the objective function and constraints (linear, nonlinear, convex), the number of variables (large or small), the smoothness of the functions (differentiable or nondifferentiable), and so on. An important distinction is between problems that have constraints on the variables and those that do not. This book is divided into two parts according to this classification.

Unconstrained optimization problems, for which we have $\mathcal{E} = \mathcal{I} = \emptyset$ in (1.1), arise directly in many practical applications. Even for some problems with natural constraints on the variables, it may be safe to disregard them as they do not affect on the solution and do not interfere with algorithms. Unconstrained problems arise also as reformulations of constrained optimization problems, in which the constraints are replaced by penalization terms added to objective function that have the effect of discouraging constraint violations.

Constrained optimization problems arise from models in which constraints play an essential role, for example in imposing budgetary constraints in an economic problem or shape constraints in a design problem. These constraints may be simple bounds such as $0 \leq x_1 \leq 100$, more general linear constraints such as $\sum_i x_i \leq 1$, or nonlinear inequalities that represent complex relationships among the variables.

When the objective function and all the constraints are linear functions of x, the problem is a *linear programming* problem. Problems of this type are probably the most widely formulated and solved of all optimization problems, particularly in management, financial, and economic applications. *Nonlinear programming* problems, in which at least some of the constraints or the objective are nonlinear functions, tend to arise naturally in the physical sciences and engineering, and are becoming more widely used in management and economic sciences as well.

GLOBAL AND LOCAL OPTIMIZATION

Many algorithms for nonlinear optimization problems seek only a local solution, a point at which the objective function is smaller than at all other feasible nearby points. They do not always find the *global solution*, which is the point with lowest function value among *all* feasible points. Global solutions are needed in some applications, but for many problems they

are difficult to recognize and even more difficult to locate. For *convex programming* problems, and more particularly for linear programs, local solutions are also global solutions. General nonlinear problems, both constrained and unconstrained, may possess local solutions that are not global solutions.

In this book we treat global optimization only in passing and focus instead on the computation and characterization of local solutions. We note, however, that many successful global optimization algorithms require the solution of many local optimization problems, to which the algorithms described in this book can be applied.

Research papers on global optimization can be found in Floudas and Pardalos [109] and in the *Journal of Global Optimization.*

STOCHASTIC AND DETERMINISTIC OPTIMIZATION

In some optimization problems, the model cannot be fully specified because it depends on quantities that are unknown at the time of formulation. This characteristic is shared by many economic and financial planning models, which may depend for example on future interest rates, future demands for a product, or future commodity prices, but uncertainty can arise naturally in almost any type of application.

Rather than just use a "best guess" for the uncertain quantities, modelers may obtain more useful solutions by incorporating additional knowledge about these quantities into the model. For example, they may know a number of possible scenarios for the uncertain demand, along with estimates of the probabilities of each scenario. *Stochastic optimization* algorithms use these quantifications of the uncertainty to produce solutions that optimize the *expected* performance of the model.

Related paradigms for dealing with uncertain data in the model include *chance-constrained optimization*, in which we ensure that the variables x satisfy the given constraints to some specified probability, and *robust optimization*, in which certain constraints are required to hold for all possible values of the uncertain data.

We do not consider stochastic optimization problems further in this book, focusing instead on *deterministic optimization* problems, in which the model is completely known. Many algorithms for stochastic optimization do, however, proceed by formulating one or more deterministic subproblems, each of which can be solved by the techniques outlined here.

Stochastic and robust optimization have seen a great deal of recent research activity. For further information on stochastic optimization, consult the books of Birge and Louveaux [22] and Kall and Wallace [174]. Robust optimization is discussed in Ben-Tal and Nemirovski [15].

CONVEXITY

The concept of convexity is fundamental in optimization. Many practical problems possess this property, which generally makes them easier to solve both in theory and practice.

The term "convex" can be applied both to sets and to functions. A set $S \in \mathbb{R}^n$ is a *convex set* if the straight line segment connecting any two points in S lies entirely inside S. Formally, for any two points $x \in S$ and $y \in S$, we have $\alpha x + (1-\alpha)y \in S$ for all $\alpha \in [0, 1]$. The function f is a *convex function* if its domain S is a convex set and if for any two points x and y in S, the following property is satisfied:

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y), \qquad \text{for all } \alpha \in [0, 1]. \tag{1.4}$$

Simple instances of convex sets include the unit ball $\{y \in \mathbb{R}^n \mid \|y\|_2 \leq 1\}$; and any polyhedron, which is a set defined by linear equalities and inequalities, that is,

$$\{x \in \mathbb{R}^n \mid Ax = b,\ Cx \leq d\},$$

where A and C are matrices of appropriate dimension, and b and d are vectors. Simple instances of convex functions include the linear function $f(x) = c^T x + \alpha$, for any constant vector $c \in \mathbb{R}^n$ and scalar α; and the convex quadratic function $f(x) = x^T H x$, where H is a symmetric positive semidefinite matrix.

We say that f is *strictly convex* if the inequality in (1.4) is strict whenever $x \neq y$ and α is in the open interval $(0, 1)$. A function f is said to be *concave* if $-f$ is convex.

If the objective function in the optimization problem (1.1) and the feasible region are both convex, then any local solution of the problem is in fact a global solution.

The term *convex programming* is used to describe a special case of the general constrained optimization problem (1.1) in which

- the objective function is convex,
- the equality constraint functions $c_i(\cdot)$, $i \in \mathcal{E}$, are linear, and
- the inequality constraint functions $c_i(\cdot)$, $i \in \mathcal{I}$, are concave.

OPTIMIZATION ALGORITHMS

Optimization algorithms are iterative. They begin with an initial guess of the variable x and generate a sequence of improved estimates (called "iterates") until they terminate, hopefully at a solution. The strategy used to move from one iterate to the next distinguishes one algorithm from another. Most strategies make use of the values of the objective function f, the constraint functions c_i, and possibly the first and second derivatives of these functions. Some algorithms accumulate information gathered at previous iterations, while others use only local information obtained at the current point. Regardless of these specifics (which will receive plenty of attention in the rest of the book), good algorithms should possess the following properties:

- Robustness. They should perform well on a wide variety of problems in their class, for all reasonable values of the starting point.

- Efficiency. They should not require excessive computer time or storage.
- Accuracy. They should be able to identify a solution with precision, without being overly sensitive to errors in the data or to the arithmetic rounding errors that occur when the algorithm is implemented on a computer.

These goals may conflict. For example, a rapidly convergent method for a large unconstrained nonlinear problem may require too much computer storage. On the other hand, a robust method may also be the slowest. Tradeoffs between convergence rate and storage requirements, and between robustness and speed, and so on, are central issues in numerical optimization. They receive careful consideration in this book.

The mathematical theory of optimization is used both to characterize optimal points and to provide the basis for most algorithms. It is not possible to have a good understanding of numerical optimization without a firm grasp of the supporting theory. Accordingly, this book gives a solid (though not comprehensive) treatment of optimality conditions, as well as convergence analysis that reveals the strengths and weaknesses of some of the most important algorithms.

NOTES AND REFERENCES

Optimization traces its roots to the calculus of variations and the work of Euler and Lagrange. The development of linear programming n the 1940s broadened the field and stimulated much of the progress in modern optimization theory and practice during the past 60 years.

Optimization is often called *mathematical programming*, a somewhat confusing term coined in the 1940s, before the word "programming" became inextricably linked with computer software. The original meaning of this word (and the intended one in this context) was more inclusive, with connotations of algorithm design and analysis.

Modeling will not be treated extensively in the book. It is an essential subject in its own right, as it makes the connection between optimization algorithms and software on the one hand, and applications on the other hand. Information about modeling techniques for various application areas can be found in Dantzig [86], Ahuja, Magnanti, and Orlin [1], Fourer, Gay, and Kernighan [112], Winston [308], and Rardin [262].

5. The Simplex Method

In this chapter we explain the simplex method for solving linear programs. We will make use of the terms *equational form* and *basic feasible solution* from the previous chapter.

Gaussian elimination in linear algebra has a fundamental theoretical and didactic significance, as a starting point for further developments. But in practice it has mostly been replaced by more complicated and more efficient algorithms. Similarly, the basic version of the simplex method that we discuss here is not commonly used for solving linear programs in practice. We do not put emphasis on the most efficient possible organization of the computations, but rather we concentrate on the main ideas.

5.1 An Introductory Example

We will first show the simplex method in action on a small concrete example, namely, on the following linear program:

$$\begin{array}{lrcl} \text{Maximize} & x_1 + x_2 & & \\ \text{subject to} & -x_1 + x_2 & \leq & 1 \\ & x_1 & \leq & 3 \\ & x_2 & \leq & 2 \\ & x_1, x_2 \geq 0. & & \end{array} \tag{5.1}$$

We intentionally do not take a linear program in equational form: The variables are nonnegative, but the inequalities have to be replaced by equations, by introducing slack variables. The equational form is

$$\begin{array}{lrcl} \text{maximize} & x_1 + x_2 & & \\ \text{subject to} & -x_1 + x_2 + x_3 & = & 1 \\ & x_1 + x_4 & = & 3 \\ & x_2 + x_5 & = & 2 \\ & x_1, x_2, \ldots, x_5 \geq 0, & & \end{array}$$

with the matrix

$$A = \begin{pmatrix} -1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

In the simplex method we first express each linear program in the form of a *simplex tableau*. In our case we begin with the tableau

$$\begin{array}{rcrcrcr} x_3 & = & 1 & + & x_1 & - & x_2 \\ x_4 & = & 3 & - & x_1 & & \\ x_5 & = & 2 & & & - & x_2 \\ \hline z & = & & & x_1 & + & x_2 \end{array}$$

The first three rows consist of the equations of the linear program, in which the slack variables have been carried over to the left-hand side and the remaining terms are on the right-hand side. The last row, separated by a line, contains a new variable z, which expresses the objective function.

Each simplex tableau is associated with a certain basic feasible solution. In our case we substitute 0 for the variables x_1 and x_2 from the right-hand side, and without calculation we see that $x_3 = 1, x_4 = 3, x_5 = 2$. This feasible solution is indeed basic with $B = \{3, 4, 5\}$; we note that A_B is the identity matrix. The variables x_3, x_4, x_5 from the left-hand side are basic and the variables x_1, x_2 from the right-hand side are nonbasic. The value of the objective function $z = 0$ corresponding to this basic feasible solution can be read off from the last row of the tableau.

From the initial simplex tableau we will construct a sequence of tableaus of a similar form, by gradually rewriting them according to certain rules. Each tableau will contain the *same* information about the linear program, only written differently. The procedure terminates with a tableau that represents the information so that the desired optimal solution can be read off directly.

Let us go to the first step. We try to increase the value of the objective function by increasing one of the nonbasic variables x_1 or x_2. In the above tableau we observe that increasing the value of x_1 (i.e. making x_1 positive) increases the value of z. The same is true for x_2, because both variables have positive coefficients in the z-row of the tableau. We can choose either x_1 or x_2; let us decide (arbitrarily) for x_2. We will increase it, while x_1 will stay 0.

By how much can we increase x_2? If we want to maintain feasibility, we have to be careful not to let any of the basic variables x_3, x_4, x_5 go below zero. This means that the equations determining x_3, x_4, x_5 may limit the increment of x_2. Let us consider the first equation

$$x_3 = 1 + x_1 - x_2.$$

Together with the implicit constraint $x_3 \geq 0$ it lets us increase x_2 up to the value $x_2 = 1$ (while keeping $x_1 = 0$). The second equation

$$x_4 = 3 - x_1$$

does not limit the increment of x_2 at all, and the third equation

$$x_5 = 2 - x_2$$

allows for an increase of x_2 up to $x_2 = 2$ before x_5 gets negative. The most stringent restriction thus follows from the first equation.

We increase x_2 as much as we can, obtaining $x_2 = 1$ and $x_3 = 0$. From the remaining equations of the tableau we get the values of the other variables:

$$\begin{aligned} x_4 &= 3 - x_1 = 3 \\ x_5 &= 2 - x_2 = 1. \end{aligned}$$

In this new feasible solution x_3 became zero and x_2 nonzero. Quite naturally we thus transfer x_3 to the right-hand side, where the nonbasic variables live, and x_2 to the left-hand side, where the basic variables reside. We do it by means of the most stringent equation $x_3 = 1 + x_1 - x_2$, from which we express

$$x_2 = 1 + x_1 - x_3.$$

We substitute the right-hand side for x_2 into the remaining equations, and we arrive at a new tableau:

$$\begin{array}{rcrcrcr} x_2 & = & 1 & + & x_1 & - & x_3 \\ x_4 & = & 3 & - & x_1 & & \\ x_5 & = & 1 & - & x_1 & + & x_3 \\ \hline z & = & 1 & + & 2x_1 & - & x_3 \end{array}$$

Here $B = \{2, 4, 5\}$, which corresponds to the basic feasible solution $\mathbf{x} = (0, 1, 0, 3, 1)$ with the value of the objective function $z = 1$.

This process of rewriting one simplex tableau into another is called a **pivot step**. In each pivot step some nonbasic variable, in our case x_2, *enters* the basis, while some basic variable, in our case x_3, *leaves* the basis.

In the new tableau we can further increase the value of the objective function by increasing x_1, while increasing x_3 would lead to a smaller z-value. The first equation does not restrict the increment of x_1 in any way, from the second one we get $x_1 \leq 3$, and from the third one $x_1 \leq 1$, so the strictest limitation is implied by the third equation. Similarly as in the previous step, we express x_1 from it and we substitute this expression into the remaining equations. Thereby x_1 enters the basis and moves to the left-hand side, and x_5 leaves the basis and migrates to the right-hand side. The tableau we obtain is

$$\begin{array}{rcrcrcr} x_1 & = & 1 & + & x_3 & - & x_5 \\ x_2 & = & 2 & & & - & x_5 \\ x_4 & = & 2 & - & x_3 & + & x_5 \\ \hline z & = & 3 & + & x_3 & - & 2x_5 \end{array}$$

with $B = \{1, 2, 4\}$, basic feasible solution $\mathbf{x} = (1, 2, 0, 2, 0)$, and $z = 3$. After one more pivot step, in which x_3 enters the basis and x_4 leaves it, we arrive at the tableau

$$\begin{array}{rcrcrcr} x_1 & = & 3 & - & x_4 & & \\ x_2 & = & 2 & & & - & x_5 \\ x_3 & = & 2 & - & x_4 & + & x_5 \\ \hline z & = & 5 & - & x_4 & - & x_5 \end{array}$$

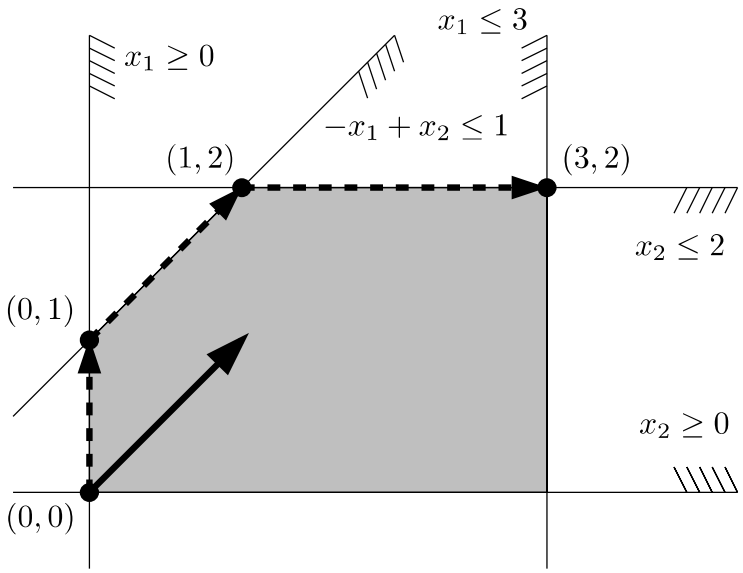
with basis $\{1, 2, 3\}$, basic feasible solution $\mathbf{x} = (3, 2, 2, 0, 0)$, and $z = 5$. In this tableau, no nonbasic variable can be increased without making the objective function value smaller, so we are stuck. Luckily, this also means that we have already found an optimal solution! Why?

Let us consider an *arbitrary* feasible solution $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_5)$ of our linear program, with the objective function attaining some value $\tilde{z}$. Now $\tilde{\mathbf{x}}$ and $\tilde{z}$ satisfy all equations in the final tableau, which was obtained from the original equations of the linear program by equivalent transformations. Hence we necessarily have

$$\tilde{z} \ = 5 - \tilde{x}_4 - \tilde{x}_5.$$

Together with the nonnegativity constraints $\tilde{x}_4, \tilde{x}_5 \geq 0$ this implies $\tilde{z} \leq 5$. The tableau even delivers a proof that $\mathbf{x} = (3, 2, 2, 0, 0)$ is the *only* optimal solution: If $z = 5$, then $x_4 = x_5 = 0$, and this determines the values of the remaining variables uniquely.

A geometric illustration. For each feasible solution (x_1, x_2) of the original linear program (5.1) with inequalities we have exactly one corresponding feasible solution $(x_1, x_2, \ldots, x_5)$ of the modified linear program in equational form, and conversely. The sets of feasible solutions are isomorphic in a suitable sense, and we can thus follow the progress of the simplex method narrated above in a planar picture for the original linear program (5.1):



We can see the simplex method moving along the edges from one feasible solution to another, while the value of the objective function grows until it reaches the optimum. In the example we could also take a shorter route if we decided to increase x_1 instead of x_2 in the first step.

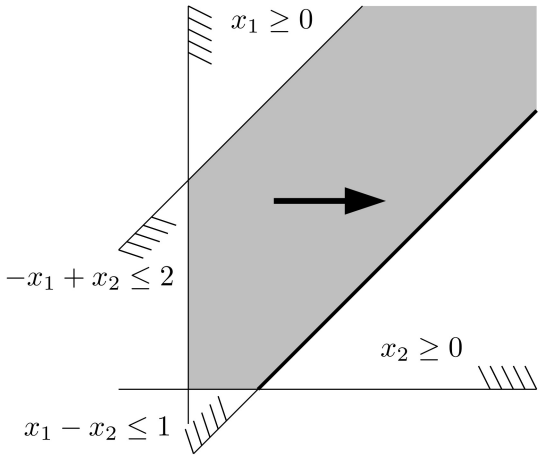
Potential troubles. In our modest example the simplex method has run smoothly without any problems. In general we must deal with several complications. We will demonstrate them on examples in the next sections.

5.2 Exception Handling: Unboundedness

What happens in the simplex method for an unbounded linear program? We will show it on the example

$$\begin{array}{ll} \text{maximize} & x_1 \\ \text{subject to} & x_1 - x_2 \leq 1 \\ & -x_1 + x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{array}$$

illustrated in the picture below:



After the usual transformation to equational form by introducing slack variables x_3, x_4, we can use these variables as a feasible basis and we obtain the initial simplex tableau

$$\begin{array}{rcrcrcr} x_3 & = & 1 & - & x_1 & + & x_2 \\ x_4 & = & 2 & + & x_1 & - & x_2 \\ \hline z & = & & & x_1 & & \end{array}$$

After the first pivot step with entering variable x_1 and leaving variable x_3 the next tableau is

$$\begin{array}{rcrcrcr} x_1 & = & 1 & + & x_2 & - & x_3 \\ x_4 & = & 3 & & & - & x_3 \\ \hline z & = & 1 & + & x_2 & - & x_3 \end{array}$$

If we now try to introduce x_2 into the basis, we discover that none of the equations in the tableau restrict its increase in any way. We can thus take x_2 arbitrarily large, and we also get z arbitrarily large—the linear program is unbounded.

Let us analyze this situation in more detail. From the tableau one can see that for an arbitrarily large number $t \geq 0$ we obtain a feasible solution by setting $x_2 = t$, $x_3 = 0$, $x_1 = 1 + t$, and $x_4 = 3$, with the value of the objective function $z = 1 + t$. In other words, the semi-infinite ray

$$\{(1, 0, 0, 3) + t(1, 1, 0, 0) : t \geq 0\}$$

is contained in the set of feasible solutions. It "witnesses" the unboundedness of the linear program, since the objective function attains arbitrarily large values on it. The corresponding semi-infinite ray for the original two-dimensional linear program is drawn thick in the picture above.

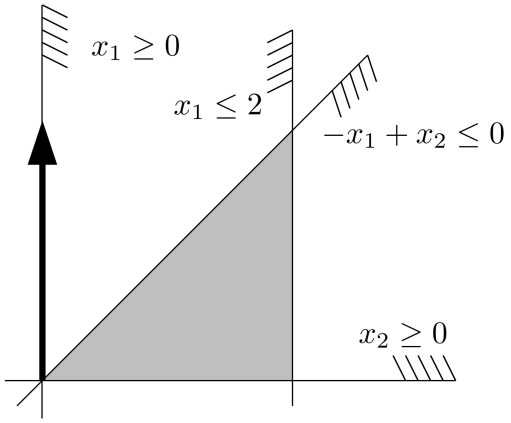
A similar ray is the output of the simplex method for all unbounded linear programs.

5.3 Exception Handling: Degeneracy

While we can make some nonbasic variable arbitrarily large in the unbounded case, the other extreme happens in a situation called a degeneracy: The equations in a tableau do not permit any increment of the selected nonbasic variable, and it may actually be impossible to increase the objective function z in a single pivot step.

Let us consider the linear program

$$\begin{array}{lrcl} \text{maximize} & x_2 & & \\ \text{subject to} & -x_1 + x_2 & \leq & 0 \\ & x_1 & \leq & 2 \\ & x_1, x_2 \geq 0. & & \end{array} \tag{5.2}$$



In the usual way we convert it to equational form and construct the initial tableau

$$\begin{array}{rcrcr} x_3 & = & & x_1 & - x_2 \\ x_4 & = & 2 & - x_1 & \\ \hline z & = & & & x_2 \end{array}$$

The only candidate for entering the basis is x_2, but the first row of the tableau shows that its value cannot be increased without making x_3 negative. Unfortunately, the impossibility of making progress in this case does not imply optimality, so we have to perform a degenerate pivot step, i.e., one with zero progress in the objective function. In our example, bringing x_2 into

the basis (with x_3 leaving) results in another tableau with the same basic feasible solution $(0, 0, 0, 2)$:

$$\begin{array}{rcrcr} x_2 & = & & x_1 & - x_3 \\ x_4 & = & 2 - & x_1 & \\ \hline z & = & & x_1 & - x_3 \end{array}$$

Nevertheless, the situation has improved. The nonbasic variable x_1 can now be increased, and by entering it into the basis (replacing x_4) we already obtain the final tableau

$$\begin{array}{rcrcr} x_1 & = & 2 & & - x_4 \\ x_2 & = & 2 - & x_3 & - x_4 \\ \hline z & = & 2 - & x_3 & - x_4 \end{array}$$

with an optimal solution $\mathbf{x} = (2, 2, 0, 0)$.

A situation that forces a degenerate pivot step may occur only for a linear program in which several feasible bases correspond to a single basic feasible solution. Such linear programs are called **degenerate**.

It is easily seen that in order that a single basic feasible solution be obtained from several bases, some of the *basic* variables have to be zero.

In this example, after one degenerate pivot step we could again make progress. In general, there might be longer runs of degenerate pivot steps. It may even happen that some tableau is repeated in a sequence of degenerate pivot steps, and so the algorithm might pass through an infinite sequence of tableaus without any progress. This phenomenon is called **cycling**. An example of a linear program for which the simplex method may cycle can be found in Chvátal's textbook cited in Chapter 9 (the smallest possible example has 6 variables and 3 equations), and we will not present it here.

If the simplex method doesn't cycle, then it necessarily finishes in a finite number of steps. This is because there are only finitely many possible simplex tableaus for any given linear program, namely at most $\binom{n}{m}$, which we will prove in Section 5.5.

How can cycling be prevented? This is a nontrivial issue and it will be discussed in Section 5.8.

5.4 Exception Handling: Infeasibility

In order that the simplex method be able to start at all, we need a feasible basis. In examples discussed up until now we got a feasible basis more or less for free. It works this way for all linear programs of the form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} \leq \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}$$

with $\mathbf{b} \geq \mathbf{0}$. Indeed, the indices of the slack variables introduced in the transformation to equational form can serve as a feasible basis.

However, in general, finding any feasible solution of a linear program is equally as difficult as finding an optimal solution (see the remark in Section 1.3). But computing the initial feasible basis can be done by the simplex method itself, if we apply it to a suitable auxiliary problem.

Let us consider the linear program in equational form

$$\begin{array}{lrcl} \text{maximize} & x_1 + 2x_2 & & \\ \text{subject to} & x_1 + 3x_2 + x_3 & = & 4 \\ & 2x_2 + x_3 & = & 2 \\ & x_1, x_2, x_3 \geq 0. & & \end{array}$$

Let us try to produce a feasible solution starting with $(x_1, x_2, x_3) = (0, 0, 0)$. This vector is nonnegative, but of course it is not feasible, since it does not satisfy the equations of the linear program. We introduce auxiliary variables x_4 and x_5 as "corrections" of infeasibility: $x_4 = 4 - x_1 - 3x_2 - x_3$ expresses by how much the original variables x_1, x_2, x_3 fail to satisfy the first equation, and $x_5 = 2 - 2x_2 - x_3$ plays a similar role for the second equation. If we managed to find nonnegative values of x_1, x_2, x_3 for which both of these corrections come out as zeros, we would have a feasible solution of the considered linear program.

The task of finding nonnegative x_1, x_2, x_3 with zero corrections can be captured by a linear program:

$$\begin{array}{lrcl} \text{Maximize} & -x_4 - x_5 & & \\ \text{subject to} & x_1 + 3x_2 + x_3 + x_4 & = & 4 \\ & 2x_2 + x_3 + x_5 & = & 2 \\ & x_1, x_2, \ldots, x_5 \geq 0. & & \end{array}$$

The optimal value of the objective function $-x_4 - x_5$ is 0 exactly if there exist values of x_1, x_2, x_3 with zero corrections, i.e., a feasible solution of the original linear program.

This is the right auxiliary linear program. The variables x_4 and x_5 form a feasible basis, with the basic feasible solution $(0, 0, 0, 4, 2)$. (Here we use that the right-hand sides, 4 and 2, are nonnegative, but since we deal with *equations*, this can always be achieved by sign changes.) Once we express the objective function using the nonbasic variables, that is, in the form $z = -6 + x_1 + 5x_2 + 2x_3$, we can start the simplex method on the auxiliary linear program.

The auxiliary linear program is surely bounded, since the objective function cannot be positive. The simplex method thus computes a basic feasible solution that is optimal.

As training the reader can check that if we let x_1 enter the basis in the first pivot step and x_3 in the second, the final simplex tableau comes out as

$$\begin{array}{rcl} x_1 & = & 2 - x_2 - x_4 + x_5 \\ x_3 & = & 2 - 2x_2 - x_5 \\ \hline z & = & -x_4 - x_5. \end{array}$$

The corresponding optimal solution $(2, 0, 2, 0, 0)$ yields a basic feasible solution of the original linear program: $(x_1, x_2, x_3) = (2, 0, 2)$. The initial simplex tableau for the original linear program can even be obtained from the final tableau of the auxiliary linear program, by leaving out the columns of the auxiliary variables x_4 and x_5,[1] and by changing the objective function back to the original one, expressed in terms of the nonbasic variables:

$$\begin{array}{rcrcr} x_1 & = & 2 & - & x_2 \\ x_3 & = & 2 & - & 2x_2 \\ \hline z & = & 2 & + & x_2 \end{array}$$

Starting from this tableau, a single pivot step already reaches the optimum.

5.5 Simplex Tableaus in General

In this section and the next one we formulate in general, and mostly with proofs, what has previously been explained on examples.

Let us consider a general linear program in equational form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}.$$

The simplex method applied to it computes a sequence of simplex tableaus. Each of them corresponds to a feasible basis B and it determines a basic feasible solution, as we will soon verify. (Let us recall that a feasible basis is an m-element set $B \subseteq \{1, 2, \ldots, n\}$ such that the matrix A_B is nonsingular and the (unique) solution of the system $A_B\mathbf{x}_B = \mathbf{b}$ is nonnegative.)

Formally, we will define a simplex tableau as a certain system of linear equations of a special form, in which the basic variables and the variable z, representing the value of the objective function, stand on the left-hand side and they are expressed in terms of the nonbasic variables.

A **simplex tableau** $\mathcal{T}(B)$ determined by a feasible basis B is a system of $m+1$ linear equations in variables $x_1, x_2, \ldots, x_n$, and z that has the *same set of solutions* as the system $A\mathbf{x} = \mathbf{b}$, $z = \mathbf{c}^T\mathbf{x}$, and in matrix notation looks as follows:

$$\begin{array}{rcrcr} \mathbf{x}_B & = & \mathbf{p} & + & Q\,\mathbf{x}_N \\ \hline z & = & z_0 & + & \mathbf{r}^T\mathbf{x}_N \end{array}$$

where $\mathbf{x}_B$ is the vector of the basic variables, $N = \{1, 2, \ldots, n\} \setminus B$, $\mathbf{x}_N$ is the vector of nonbasic variables, $\mathbf{p} \in \mathbb{R}^m$, $\mathbf{r} \in \mathbb{R}^{n-m}$, Q is an $m \times (n-m)$ matrix, and $z_0 \in \mathbb{R}$.

[1] It may happen that some auxiliary variables are zero but still basic in the final tableau of the auxiliary program, and so they cannot simply be left out. Section 5.6 discusses this (easy) issue.

The basic feasible solution corresponding to this tableau can be read off immediately: It is obtained by substituting $\mathbf{x}_N = \mathbf{0}$; that is, we have $\mathbf{x}_B = \mathbf{p}$. From the feasibility of the basis B we see that $\mathbf{p} \geq \mathbf{0}$. The objective function for this basic feasible solution has value $z_0 + \mathbf{r}^T\mathbf{0} = z_0$.

The values of $\mathbf{p}, Q, \mathbf{r}, z_0$ can easily be expressed using B and $A, \mathbf{b}, \mathbf{c}$:

5.5.1 Lemma. *For each feasible basis B there exists exactly one simplex tableau, and it is given by*

$$Q = -A_B^{-1}A_N,\ \mathbf{p} = A_B^{-1}\mathbf{b},\ z_0 = \mathbf{c}_B^T A_B^{-1}\mathbf{b},\quad \textit{and}\quad \mathbf{r} = \mathbf{c}_N - (\mathbf{c}_B^T A_B^{-1} A_N)^T.$$

It is neither necessary nor very useful to remember these formulas; they are easily rederived if needed. The proof is not very exciting and we write it more concisely than other parts of the text and we leave some details to a diligent reader. We will proceed similarly with subsequent proofs of a similar kind.

Proof. First let us see how these formulas can be discovered: We rewrite the system $A\mathbf{x} = \mathbf{b}$ to $A_B\mathbf{x}_B = \mathbf{b} - A_N\mathbf{x}_N$, and we multiply it by the inverse matrix A_B^{-1} from the left (these transformations preserve the solution set), which leads to

$$\mathbf{x}_B = A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N.$$

We substitute the right-hand side for $\mathbf{x}_B$ into the equation $z = \mathbf{c}^T\mathbf{x} = \mathbf{c}_B^T\mathbf{x}_B + \mathbf{c}_N^T\mathbf{x}_N$, and we obtain

$$\begin{aligned} z &= \mathbf{c}_B^T(A_B^{-1}\mathbf{b} - A_B^{-1}A_N\mathbf{x}_N) + \mathbf{c}_N^T\mathbf{x}_N \\ &= \mathbf{c}_B^T A_B^{-1}\mathbf{b} + (\mathbf{c}_N^T - c_B^T A_B^{-1}A_N)\mathbf{x}_N. \end{aligned}$$

Thus the formulas in the lemma do yield a simplex tableau, and it remains to verify the uniqueness.

Let $\mathbf{p}, Q, \mathbf{r}, z_0$ determine a simplex tableau for a feasible basis B, and let $\mathbf{p}', Q', \mathbf{r}', z_0'$ do as well. Since each choice of $\mathbf{x}_N$ determines $\mathbf{x}_B$ uniquely, the equality $\mathbf{p} + Q\mathbf{x}_N = \mathbf{p}' + Q'\mathbf{x}_N$ has to hold for all $\mathbf{x}_N \in \mathbb{R}^{n-m}$. The choice $\mathbf{x}_N = \mathbf{0}$ gives $\mathbf{p} = \mathbf{p}'$, and if we substitute the unit vectors $\mathbf{e}_j$ of the standard basis for $\mathbf{x}_N$ one by one, we also get $Q = Q'$. The equalities $z_0 = z_0'$ and $\mathbf{r} = \mathbf{r}'$ are proved similarly. □

5.6 The Simplex Method in General

Optimality. Exactly as in the concrete example in Section 5.1, we have the following criterion of optimality of a simplex tableau:

If $\mathcal{T}(B)$ is a simplex tableau such that the coefficients of the nonbasic variables are nonpositive in the last row, i.e., if

$$\mathbf{r} \leq \mathbf{0},$$

then the corresponding basic feasible solution is *optimal*.

Indeed, the basic feasible solution corresponding to such a tableau has the objective function equal to z_0, while for any other feasible solution $\tilde{\mathbf{x}}$ we have $\tilde{\mathbf{x}}_N \geq 0$ and $\mathbf{c}^T\tilde{\mathbf{x}} = z_0 + \mathbf{r}^T\tilde{\mathbf{x}}_N \leq z_0$.

A pivot step: who enters and who leaves. In each step of the simplex method we go from an "old" basis B and simplex tableau $\mathcal{T}(B)$ to a "new" basis B' and the corresponding simplex tableau $\mathcal{T}(B')$. A nonbasic variable x_v enters the basis and a basic variable x_u leaves the basis,[2] and hence $B' = (B \setminus \{u\}) \cup \{v\}$.

We always select the entering variable x_v first.

A nonbasic variable may enter the basis if and only if its coefficient in the last row of the simplex tableau is *positive*.

Only incrementing such nonbasic variables increases the value of the objective function.

Usually there are several positive coefficients in the last row, and hence several possible choices of the entering variable. For the time being the reader may think of this choice as arbitrary. We will discuss ways of selecting one of these possibilities in Section 5.7.

Once we decide that the entering variable is some x_v, it remains to pick the leaving variable.

The leaving variable x_u has to be such that its nonnegativity, together with the corresponding equation in the simplex tableau having x_u on the left-hand side, limits the increment of the entering variable x_v most strictly.

Expressed by a formula, this condition might look complicated because of some double indices, but the idea is simple and we have already seen it in examples. Let us write $B = \{k_1, k_2, \ldots, k_m\}$, $k_1 < k_2 < \cdots < k_m$, and $N = \{\ell_1, \ell_2, \ldots, \ell_{n-m}\}$, $\ell_1 < \ell_2 < \cdots < \ell_{n-m}$. Then the ith equation of the simplex tableau has the form

$$x_{k_i} = p_i + \sum_{j=1}^{n-m} q_{ij} x_{\ell_j}.$$

[2] The letters u and v do not denote vectors here (the alphabet is not that long, after all).

We now want to write the index v of the chosen entering variable as $v = \ell_\beta$. In more detail, we define $\beta \in \{1, 2, \ldots, n-m\}$ as the index for which $v = \ell_\beta$. Similarly, the index u of the leaving variable (which hasn't been selected yet) will be written in the form $u = k_\alpha$.

Since all nonbasic variables x_{ℓ_j}, $j \neq \beta$, should remain zero, the nonnegativity condition $x_{k_i} \geq 0$ limits the possible values of the entering variable x_{ℓ_β} by the inequality $-q_{i\beta}x_{\ell_\beta} \leq p_i$. If $q_{i\beta} \geq 0$, then this inequality doesn't restrict the increase of x_{ℓ_β} in any way, while for $q_{i\beta} < 0$ it yields the restriction $x_{\ell_\beta} \leq -p_i/q_{i\beta}$.

The leaving variable x_{k_α} is thus always such that

$$q_{\alpha\beta} < 0 \quad \text{and} \quad -\frac{p_\alpha}{q_{\alpha\beta}} = \min\left\{-\frac{p_i}{q_{i\beta}} : q_{i\beta} < 0,\ i = 1, 2, \ldots, m\right\}. \tag{5.3}$$

That is, in the simplex tableau we consider only the rows in which the coefficient of x_v is negative. In such rows we divide by this coefficient the component of the vector $\mathbf{p}$, we change sign, and we seek a minimum among these ratios. If there is no row with a negative coefficient of x_v, i.e., the minimum of the right-hand side of equation (5.3) is over an empty set, then the linear program is unbounded and the computation finishes.

For a proof that the simplex method really goes through a sequence of feasible bases we need the following lemma.

5.6.1 Lemma. *If B is a feasible basis and $\mathcal{T}(B)$ is the corresponding simplex tableau, and if the entering variable x_v and the leaving variable x_u have been selected according to the criteria described above (and otherwise arbitrarily), then $B' = (B \setminus \{u\}) \cup \{v\}$ is again a feasible basis.*

If no x_u satisfies the criterion for a leaving variable, then the linear program is unbounded. For all $t \geq 0$ we obtain a feasible solution by substituting t for x_v and 0 for all other nonbasic variables, and the value of the objective function for these feasible solutions tends to infinity as $t \to \infty$.

The proof is one of those not essential for a basic understanding of the material.

Proof (sketch). We first need to verify that the matrix $A_{B'}$ is nonsingular. This holds exactly if $A_B^{-1}A_{B'}$ is nonsingular, since we assume nonsingularity of A_B. The matrix $A_{B'}$ agrees with A_B in $m-1$ columns corresponding to the basic variable indices $B \setminus \{u\}$. For the basic variable with index $k_i, i \neq \alpha$, we get the unit vector $\mathbf{e}_i$, in the corresponding column of $A_B^{-1}A_{B'}$.

The negative of the remaining column of the matrix $A_B^{-1}A_{B'}$ occurs in the simplex tableau $\mathcal{T}(B)$ as the column of the entering variable x_v, since $Q = -A_B^{-1}A_N$ by Lemma 5.5.1. There is a nonzero number $q_{\alpha\beta}$ in row α corresponding to the leaving variable x_u, since we have selected

x_u that way, and the other columns of $A_B^{-1}A_{B'}$ have 0 in that row. Hence the matrix is nonsingular as claimed.

Next, we need to check feasibility of the basis B'. Here we use the fact that the new basic feasible solution, that for B', can be written in terms of the old one, and the nonnegativity of its basic variables are exactly those conditions that are used for choosing the leaving variable.

In practically the same way one can show the part of the lemma dealing with unbounded linear programs. We omit further details. □

A geometric view. As we saw in Section 4.4, basic feasible solutions are vertices of the polyhedron of feasible solutions. It is not hard to verify that a pivot step of the simplex method corresponds to a move from one vertex to another along an edge of the polyhedron (where an edge is a 1-dimensional face, i.e., a segment connecting the considered vertices; see Section 4.4).

Degenerate pivot steps are an exception, where we stay at the same vertex and only the feasible basis changes. A vertex of an n-dimensional convex polyhedron is generally determined by n of the bounding hyperplanes (think of a 3-dimensional cube, say). Degeneracy can occur only if we have more than n of the bounding hyperplanes meeting at a vertex (this happens for the 3-dimensional regular octahedron, for example).

Organization of the computations. Whenever we find a new feasible basis as above, we could compute the new simplex tableau according to the formulas from Lemma 5.5.1. But this is never done since it is inefficient.

For *hand calculation* the new simplex tableau is computed from the old one. We have already illustrated one possible approach in the examples. We take the equation of the old tableau with the leaving variable x_u on the left, and in this equation we carry the entering variable x_v over to the left and x_u to the right. The modified equation becomes the equation for x_v in the new tableau. The right-hand side is then substituted for x_v into all of the other equations, including the one for z in the last row. This finishes the construction of the new tableau.

In computer implementations of the simplex method, the simplex tableau is typically not computed in full. Rather, only the basic components of the basic feasible solution, i.e., the vector $\mathbf{p} = A_B^{-1}\mathbf{b}$, and the matrix A_B^{-1} are maintained. The latter allows for a fast computation of other entries of the simplex tableau when they are needed. (Let us note that for the optimality test and for selecting the entering variable we need only the last row, and for selecting the leaving variable we need only $\mathbf{p}$ and the column of the entering variable.) With respect to efficiency and numerical accuracy, the explicit inverse A_B^{-1} is not

the best choice, and in practice, it is often represented by an (approximate) LU-factorization of the matrix A_B, or by other devices that can easily be updated during a pivot step of the simplex method. Since an efficient implementation of the simplex method is not among our main concerns, we will not describe how these things are actually done.

This computational approach is called the *revised simplex method*. For m considerably smaller than n it is usually much more efficient than maintaining all of the simplex tableau. In particular, $O(m^2)$ arithmetic operations per pivot step are sufficient for maintaining an LU-factorization of A_B, as opposed to about mn operations required for maintaining the simplex tableau.

Computing an initial feasible basis. If the given linear program has no "obvious" feasible basis, we look for an initial feasible basis by the procedure indicated in Section 5.4. For a linear program in the usual equational form

$$\text{maximize } \mathbf{c}^T\mathbf{x} \text{ subject to } A\mathbf{x} = \mathbf{b} \text{ and } \mathbf{x} \geq \mathbf{0}$$

we first arrange for $\mathbf{b} \geq \mathbf{0}$: We multiply the equations with $b_i < 0$ by -1. Then we introduce m new variables x_{n+1} through x_{n+m}, and we solve the auxiliary linear program

$$\begin{array}{ll} \text{maximize} & -(x_{n+1} + x_{n+2} + \cdots + x_{n+m}) \\ \text{subject to} & \bar{A}\,\overline{\mathbf{x}} = \mathbf{b} \\ & \overline{\mathbf{x}} \geq \mathbf{0}, \end{array}$$

where $\overline{\mathbf{x}} = (x_1, \ldots, x_{n+m})$ is the vector of all variables including the new ones, and $\bar{A} = (A \,|\, I_m)$ is obtained from A by appending the $m \times m$ identity matrix to the right. The original linear program is feasible if and only if every optimal solution of the auxiliary linear program satisfies $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$. Indeed, it is clear that an optimal solution of the auxiliary linear program with $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$ yields a feasible solution of the original linear program. Conversely, any feasible solution of the original linear program provides a feasible solution of the auxiliary linear program that has the objective function equal to 0 and is thus optimal.

The auxiliary linear program can be solved by the simplex method directly, since the new variables x_{n+1} through x_{n+m} constitute an initial feasible basis. In this way we obtain some optimal solution. If it doesn't satisfy $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$, we are done—the original linear program is infeasible.

Let us assume that the optimal solution of the auxiliary linear program has $x_{n+1} = x_{n+2} = \cdots = x_{n+m} = 0$. The simplex method always returns a basic feasible solution. If none of the new variables x_{n+1} through x_{n+m} are in the basis for the returned optimal solution, then such a basis is then a feasible basis for the original linear program, too, and it allows us to start the simplex method.

In some degenerate cases it may happen that the basis returned by the simplex method for the auxiliary linear program contains some of the variables $x_{n+1}, \ldots, x_{n+m}$, and such a basis cannot directly be used for the original linear program. But this is a cosmetic problem only: From the returned optimal solution one can get a feasible basis for the original linear program by simple linear algebra. Namely, the optimal solution has at most m nonzero components, and their columns in the matrix A are linearly independent. If these columns are fewer than m, we can add more linearly independent columns and thus get a basis; see the proof of Lemma 4.2.1.

5.7 Pivot Rules

A **pivot rule** is a rule for selecting the entering variable if there are several possibilities, which is usually the case. Sometimes there may also be more than one possibility for choosing the leaving variable, and some pivot rules specify this choice as well, but this part is typically not so important.

The number of pivot steps needed for solving a linear program depends substantially on the pivot rule. (See the example in Section 5.1.) The problem is, of course, that we do not know in advance which choices will be good in the long run.

Here we list some of the common pivot rules. By an "improving variable" we mean any nonbasic variable with a positive coefficient in the z-row of the simplex tableau, in other words, a candidate for the entering variable.

LARGEST COEFFICIENT. Choose an improving variable with the largest coefficient in the row of the objective function z. This is the original rule, suggested by Dantzig, that maximizes the improvement of z *per unit increase* of the entering variable.

LARGEST INCREASE. Choose an improving variable that leads to the largest *absolute* improvement in z. This rule is computationally more expensive than the LARGEST COEFFICIENT rule, but it locally maximizes the progress.

STEEPEST EDGE. Choose an improving variable whose entering into the basis moves the current basic feasible solution in a direction closest to the direction of the vector $\mathbf{c}$. Written by a formula, the ratio

$$\frac{\mathbf{c}^T(\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{old}})}{\|\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{old}}\|}$$

should be maximized, where $\mathbf{x}_{\text{old}}$ is the basic feasible solution for the current simplex tableau and $\mathbf{x}_{\text{new}}$ is the basic feasible solution for the tableau that would be obtained by entering the considered improving variable into the basis. (We recall that $\|\mathbf{v}\| = (v_1^2 + v_2^2 + \cdots + v_n^2)^{1/2} = \sqrt{\mathbf{v}^T\mathbf{v}}$ denotes the Euclidean length of the vector $\mathbf{v}$, and the expression $\mathbf{u}^T\mathbf{v}/(\|\mathbf{u}\| \cdot \|\mathbf{v}\|)$ is the cosine of the angle of the vectors $\mathbf{u}$ and $\mathbf{v}$.)

CHAPTER 2

Fundamentals of Unconstrained Optimization

In unconstrained optimization, we minimize an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is

$$\min_x \ f(x), \tag{2.1}$$

where $x \in \mathbb{R}^n$ is a real vector with $n \geq 1$ components and $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function.

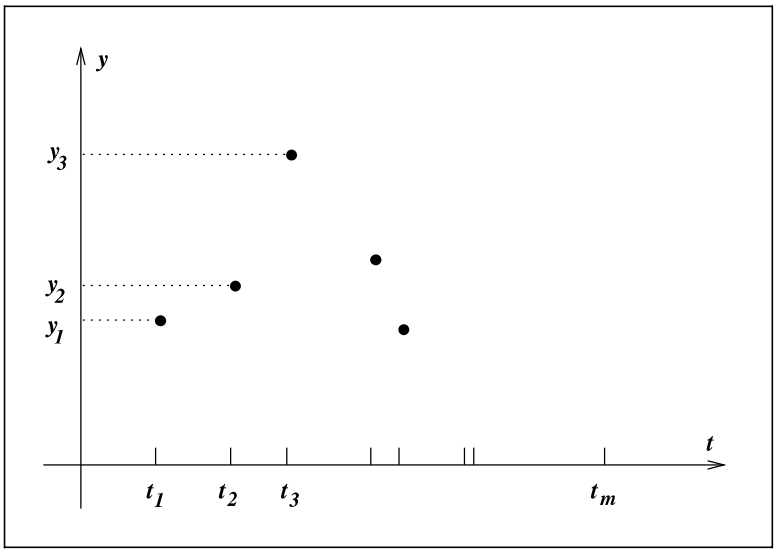


Figure 2.1 Least squares data fitting problem.

Usually, we lack a global perspective on the function f. All we know are the values of f and maybe some of its derivatives at a set of points $x_0, x_1, x_2, \ldots$. Fortunately, our algorithms get to choose these points, and they try to do so in a way that identifies a solution reliably and without using too much computer time or storage. Often, the information about f does not come cheaply, so we usually prefer algorithms that do not call for this information unnecessarily.

❑ EXAMPLE 2.1

Suppose that we are trying to find a curve that fits some experimental data. Figure 2.1 plots measurements $y_1, y_2, \ldots, y_m$ of a signal taken at times $t_1, t_2, \ldots, t_m$. From the data and our knowledge of the application, we deduce that the signal has exponential and oscillatory behavior of certain types, and we choose to model it by the function

$$\phi(t; x) = x_1 + x_2 e^{-(x_3 - t)^2/x_4} + x_5 \cos(x_6 t).$$

The real numbers x_i, $i = 1, 2, \ldots, 6$, are the parameters of the model; we would like to choose them to make the model values $\phi(t_j; x)$ fit the observed data y_j as closely as possible. To state our objective as an optimization problem, we group the parameters x_i into a vector of unknowns $x = (x_1, x_2, \ldots, x_6)^T$, and define the residuals

$$r_j(x) = y_j - \phi(t_j; x), \qquad j = 1, 2, \ldots, m, \tag{2.2}$$

which measure the discrepancy between the model and the observed data. Our estimate of

x will be obtained by solving the problem

$$\min_{x \in \mathbb{R}^6} f(x) = r_1^2(x) + r_2^2(x) + \cdots + r_m^2(x). \tag{2.3}$$

This is a *nonlinear least-squares problem*, a special case of unconstrained optimization. It illustrates that some objective functions can be expensive to evaluate even when the number of variables is small. Here we have $n = 6$, but if the number of measurements m is large (10^5, say), evaluation of $f(x)$ for a given parameter vector x is a significant computation.

❐

Suppose that for the data given in Figure 2.1 the optimal solution of (2.3) is approximately $x^* = (1.1, 0.01, 1.2, 1.5, 2.0, 1.5)$ and the corresponding function value is $f(x^*) = 0.34$. Because the optimal objective is nonzero, there must be discrepancies between the observed measurements y_j and the model predictions $\phi(t_j, x^*)$ for some (usually most) values of j—the model has not reproduced all the data points exactly. How, then, can we verify that x^* is indeed a minimizer of f? To answer this question, we need to define the term "solution" and explain how to recognize solutions. Only then can we discuss algorithms for unconstrained optimization problems.

2.1 WHAT IS A SOLUTION?

Generally, we would be happiest if we found a *global minimizer* of f, a point where the function attains its least value. A formal definition is

> A point x^* is a *global minimizer* if $f(x^*) \leq f(x)$ for all x,

where x ranges over all of $\mathbb{R}^n$ (or at least over the domain of interest to the modeler). The global minimizer can be difficult to find, because our knowledge of f is usually only local. Since our algorithm does not visit many points (we hope!), we usually do not have a good picture of the overall shape of f, and we can never be sure that the function does not take a sharp dip in some region that has not been sampled by the algorithm. Most algorithms are able to find only a *local* minimizer, which is a point that achieves the smallest value of f in its neighborhood. Formally, we say:

> A point x^* is a *local minimizer* if there is a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) \leq f(x)$ for all $x \in \mathcal{N}$.

(Recall that a neighborhood of x^* is simply an open set that contains x^*.) A point that satisfies this definition is sometimes called a *weak local minimizer*. This terminology distinguishes

it from a strict local minimizer, which is the outright winner in its neighborhood. Formally,

> A point x^* is a *strict local minimizer* (also called a *strong local minimizer*) if there is a neighborhood $\mathcal{N}$ of x^* such that $f(x^*) < f(x)$ for all $x \in \mathcal{N}$ with $x \neq x^*$.

For the constant function $f(x) = 2$, every point x is a weak local minimizer, while the function $f(x) = (x-2)^4$ has a strict local minimizer at $x = 2$.

A slightly more exotic type of local minimizer is defined as follows.

> A point x^* is an *isolated local minimizer* if there is a neighborhood $\mathcal{N}$ of x^* such that x^* is the only local minimizer in $\mathcal{N}$.

Some strict local minimizers are not isolated, as illustrated by the function

$$f(x) = x^4 \cos(1/x) + 2x^4, \qquad f(0) = 0,$$

which is twice continuously differentiable and has a strict local minimizer at $x^* = 0$. However, there are strict local minimizers at many nearby points x_j, and we can label these points so that $x_j \to 0$ as $j \to \infty$.

While strict local minimizers are not always isolated, it is true that all isolated local minimizers are strict.

Figure 2.2 illustrates a function with many local minimizers. It is usually difficult to find the global minimizer for such functions, because algorithms tend to be "trapped" at local minimizers. This example is by no means pathological. In optimization problems associated with the determination of molecular conformation, the potential function to be minimized may have millions of local minima.

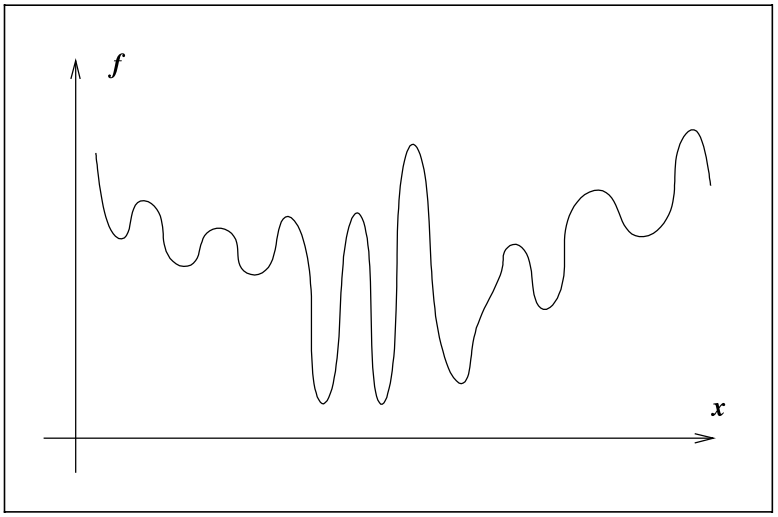


Figure 2.2 A difficult case for global minimization.

Sometimes we have additional "global" knowledge about f that may help in identifying global minima. An important special case is that of convex functions, for which every local minimizer is also a global minimizer.

RECOGNIZING A LOCAL MINIMUM

From the definitions given above, it might seem that the only way to find out whether a point x^* is a local minimum is to examine all the points in its immediate vicinity, to make sure that none of them has a smaller function value. When the function f is *smooth*, however, there are more efficient and practical ways to identify local minima. In particular, if f is twice continuously differentiable, we may be able to tell that x^* is a local minimizer (and possibly a strict local minimizer) by examining just the gradient $\nabla f(x^*)$ and the Hessian $\nabla^2 f(x^*)$.

The mathematical tool used to study minimizers of smooth functions is Taylor's theorem. Because this theorem is central to our analysis throughout the book, we state it now. Its proof can be found in any calculus textbook.

Theorem 2.1 (Taylor's Theorem).
Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and that $p \in \mathbb{R}^n$. Then we have that

$$f(x+p) = f(x) + \nabla f(x+tp)^T p, \tag{2.4}$$

for some $t \in (0,1)$. Moreover, if f is twice continuously differentiable, we have that

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+tp)p\, dt, \tag{2.5}$$

and that

$$f(x+p) = f(x) + \nabla f(x)^T p + \tfrac{1}{2} p^T \nabla^2 f(x+tp)p, \tag{2.6}$$

for some $t \in (0,1)$.

Necessary conditions for optimality are derived by assuming that x^* is a local minimizer and then proving facts about $\nabla f(x^*)$ and $\nabla^2 f(x^*)$.

Theorem 2.2 (First-Order Necessary Conditions).
If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.

PROOF. Suppose for contradiction that $\nabla f(x^*) \neq 0$. Define the vector $p = -\nabla f(x^*)$ and note that $p^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$. Because ∇f is continuous near x^*, there is a scalar $T > 0$ such that

$$p^T \nabla f(x^* + tp) < 0, \qquad \text{for all } t \in [0, T].$$

For any $\bar{t} \in (0, T]$, we have by Taylor's theorem that

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t} p^T \nabla f(x^* + tp), \qquad \text{for some } t \in (0, \bar{t}).$$

Therefore, $f(x^* + \bar{t}p) < f(x^*)$ for all $\bar{t} \in (0, T]$. We have found a direction leading away from x^* along which f decreases, so x^* is not a local minimizer, and we have a contradiction. □

We call x^* a *stationary point* if $\nabla f(x^*) = 0$. According to Theorem 2.2, any local minimizer must be a stationary point.

For the next result we recall that a matrix B is positive definite if $p^T Bp > 0$ for all $p \neq 0$, and positive semidefinite if $p^T Bp \geq 0$ for all p (see the Appendix).

Theorem 2.3 (Second-Order Necessary Conditions).
If x^* is a local minimizer of f and $\nabla^2 f$ exists and is continuous in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.

PROOF. We know from Theorem 2.2 that $\nabla f(x^*) = 0$. For contradiction, assume that $\nabla^2 f(x^*)$ is not positive semidefinite. Then we can choose a vector p such that $p^T \nabla^2 f(x^*) p < 0$, and because $\nabla^2 f$ is continuous near x^*, there is a scalar $T > 0$ such that $p^T \nabla^2 f(x^* + tp) p < 0$ for all $t \in [0, T]$.

By doing a Taylor series expansion around x^*, we have for all $\bar{t} \in (0, T]$ and some $t \in (0, \bar{t})$ that

$$f(x^* + \bar{t}p) = f(x^*) + \bar{t} p^T \nabla f(x^*) + \tfrac{1}{2}\bar{t}^2 p^T \nabla^2 f(x^* + tp) p < f(x^*).$$

As in Theorem 2.2, we have found a direction from x^* along which f is decreasing, and so again, x^* is not a local minimizer. □

We now describe *sufficient conditions*, which are conditions on the derivatives of f at the point z^* that guarantee that x^* is a local minimizer.

Theorem 2.4 (Second-Order Sufficient Conditions).

Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.

PROOF. Because the Hessian is continuous and positive definite at x^*, we can choose a radius $r > 0$ so that $\nabla^2 f(x)$ remains positive definite for all x in the open ball $\mathcal{D} = \{z \mid \|z - x^*\| < r\}$. Taking any nonzero vector p with $\|p\| < r$, we have $x^* + p \in \mathcal{D}$ and so

$$\begin{aligned} f(x^* + p) &= f(x^*) + p^T \nabla f(x^*) + \tfrac{1}{2} p^T \nabla^2 f(z) p \\ &= f(x^*) + \tfrac{1}{2} p^T \nabla^2 f(z) p, \end{aligned}$$

where $z = x^* + tp$ for some $t \in (0, 1)$. Since $z \in \mathcal{D}$, we have $p^T \nabla^2 f(z) p > 0$, and therefore $f(x^* + p) > f(x^*)$, giving the result. □

Note that the second-order sufficient conditions of Theorem 2.4 guarantee something stronger than the necessary conditions discussed earlier; namely, that the minimizer is a *strict* local minimizer. Note too that the second-order sufficient conditions are not necessary: A point x^* may be a strict local minimizer, and yet may fail to satisfy the sufficient conditions. A simple example is given by the function $f(x) = x^4$, for which the point $x^* = 0$ is a strict local minimizer at which the Hessian matrix vanishes (and is therefore not positive definite).

When the objective function is convex, local and global minimizers are simple to characterize.

Theorem 2.5.

When f is convex, any local minimizer x^* is a global minimizer of f. If in addition f is differentiable, then any stationary point x^* is a global minimizer of f.

PROOF. Suppose that x^* is a local but not a global minimizer. Then we can find a point $z \in \mathbb{R}^n$ with $f(z) < f(x^*)$. Consider the line segment that joins x^* to z, that is,

$$x = \lambda z + (1 - \lambda) x^*, \qquad \text{for some } \lambda \in (0, 1]. \tag{2.7}$$

By the convexity property for f, we have

$$f(x) \leq \lambda f(z) + (1 - \lambda) f(x^*) < f(x^*). \tag{2.8}$$

Any neighborhood $\mathcal{N}$ of x^* contains a piece of the line segment (2.7), so there will always be points $x \in \mathcal{N}$ at which (2.8) is satisfied. Hence, x^* is not a local minimizer.

For the second part of the theorem, suppose that x^* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\begin{aligned}\nabla f(x^*)^T(z-x^*) &= \frac{d}{d\lambda} f(x^*+\lambda(z-x^*))\,|_{\lambda=0} \quad \text{(see the Appendix)}\\ &= \lim_{\lambda\downarrow 0} \frac{f(x^*+\lambda(z-x^*))-f(x^*)}{\lambda}\\ &\le \lim_{\lambda\downarrow 0} \frac{\lambda f(z)+(1-\lambda)f(x^*)-f(x^*)}{\lambda}\\ &= f(z)-f(x^*) < 0.\end{aligned}$$

Therefore, $\nabla f(x^*) \neq 0$, and so x^* is not a stationary point. □

These results, which are based on elementary calculus, provide the foundations for unconstrained optimization algorithms. In one way or another, all algorithms seek a point where $\nabla f(\cdot)$ vanishes.

NONSMOOTH PROBLEMS

This book focuses on smooth functions, by which we generally mean functions whose second derivatives exist and are continuous. We note, however, that there are interesting problems in which the functions involved may be nonsmooth and even discontinuous. It is not possible in general to identify a minimizer of a general discontinuous function. If, however, the function consists of a few smooth pieces, with discontinuities between the pieces, it may be possible to find the minimizer by minimizing each smooth piece individually.

If the function is continuous everywhere but nondifferentiable at certain points, as in Figure 2.3, we can identify a solution by examing the *subgradient* or *generalized*

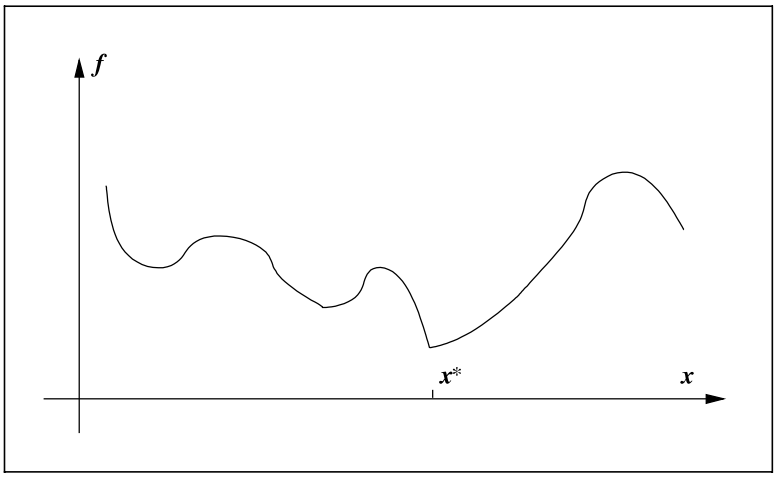


Figure 2.3 Nonsmooth function with minimum at a kink.

gradient, which are generalizations of the concept of gradient to the nonsmooth case. Nonsmooth optimization is beyond the scope of this book; we refer instead to Hiriart-Urruty and Lemaréchal [170] for an extensive discussion of theory. Here, we mention only that the minimization of a function such as the one illustrated in Figure 2.3 (which contains a jump discontinuity in the first derivative $f'(x)$ at the minimum) is difficult because the behavior of f is not predictable near the point of nonsmoothness. That is, we cannot be sure that information about f obtained at one point can be used to infer anything about f at neighboring points, because points of nondifferentiability may intervene. However, minimization of certain special nondifferentiable functions, such as

$$f(x) = \|r(x)\|_1, \qquad f(x) = \|r(x)\|_\infty \tag{2.9}$$

(where $r(x)$ is a vector function), can be reformulated as smooth constrained optimization problems; see Exercise 12.5 in Chapter 12 and (17.31). The functions (2.9) are useful in data fitting, where $r(x)$ is the residual vector whose components are defined in (2.2).

2.2 OVERVIEW OF ALGORITHMS

The last forty years have seen the development of a powerful collection of algorithms for unconstrained optimization of smooth functions. We now give a broad description of their main properties, and we describe them in more detail in Chapters 3, 4, 5, 6, and 7. All algorithms for unconstrained minimization require the user to supply a starting point, which we usually denote by x_0. The user with knowledge about the application and the data set may be in a good position to choose x_0 to be a reasonable estimate of the solution. Otherwise, the starting point must be chosen by the algorithm, either by a systematic approach or in some arbitrary manner.

Beginning at x_0, optimization algorithms generate a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ that terminate when either no more progress can be made or when it seems that a solution point has been approximated with sufficient accuracy. In deciding how to move from one iterate x_k to the next, the algorithms use information about the function f at x_k, and possibly also information from earlier iterates $x_0, x_1, \ldots, x_{k-1}$. They use this information to find a new iterate x_{k+1} with a lower function value than x_k. (There exist *nonmonotone* algorithms that do not insist on a decrease in f at every step, but even these algorithms require f to be decreased after some prescribed number m of iterations, that is, $f(x_k) < f(x_{k-m})$.)

There are two fundamental strategies for moving from the current point x_k to a new iterate x_{k+1}. Most of the algorithms described in this book follow one of these approaches.

9
Unconstrained Optimization Problems

9.1 Introduction

An optimization problem in which the decision vector x is allowed to take any value in $\mathbf{R}^n$ is called an *unconstrained optimization problem*, written in short as (UMP). This chapter is devoted to the study of certain standard algorithms for unconstrained minimization of functions of one variable as well as functions of several variables. Although most real life optimization problems are constrained optimization problems, the study of unconstrained optimization problems is important, mainly because certain efficient techniques for solving constrained optimization problems use our knowledge of solving unconstrained optimization problems.

9.2 Basic Scheme and Certain Desirable Properties

We consider the unconstrained minimization problem

$$\underset{x \in \mathbf{R}^n}{Min} \quad f(x) \tag{9.1}$$

and aim to develop algorithms for solving the same. Ideally we shall like to get a global min point of (9.1), i.e. to get a point $\bar{x} \in \mathbf{R}^n$ such that $f(\bar{x}) \leq f(x)$ for all $x \in \mathbf{R}^n$. But unfortunately this seems to be rather difficult for a general function f. Even finding a local min point, i.e. a point $\bar{x}$ such that there exists a neighbourhood $N_\delta(\bar{x})$ with $f(\bar{x}) \leq f(x)$ for all $x \in N_\delta(\bar{x})$, is also not always possible.

In view of the above, we most often decide to identify a set $\Omega \subseteq \mathbf{R}^n$, called the *solution set*, such that under certain additional conditions on the function f (e.g. convexity) any point of Ω becomes a global min point of problem (9.1).

So far we have not put any smoothness condition on the function f. Let us now assume that the function f is continuously differentiable. Sometimes we may have also to assume that f is twice continuously differentiable but this condition we shall mention specifically if needed.

Definition 9.2.1 (Solution Set). *Let $f : \mathbf{R}^n \longrightarrow \mathbf{R}$ be continuously differentiable. Then the set*

$$\Omega = \{ x \in \mathbf{R}^n : \nabla f(x) = 0 \}$$

is called the solution set of (9.1).

Remark 9.2.1 *Let $f : \mathbf{R}^n \longrightarrow \mathbf{R}$ be differentiable convex function and $\bar{x}$ be a point of the solution set. Then $\bar{x}$ is a global min point of problem (9.1).*

We now describe a common *basic scheme* of the form

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

for solving the unconstrained minimization problem (9.1) where $x^{(k)}$ is the current solution, $d^{(k)}$ is the *direction of movement* from $x^{(k)}$ and $\overline{\alpha_k} > 0$ (called the *step size*) is the distance upto which we move in the direction $d^{(k)}$ from the current point $x^{(k)}$. The obvious question now is how to find the direction for movement $d^{(k)}$ and how to determine the step size $\overline{\alpha_k}$? Various algorithms for solving problem (9.1) have been devised to determine the direction $d^{(k)}$ and the step size $\overline{\alpha_k}$ so that the sequence of iterates $\{x^{(k)}\}$ converges to a point $\bar{x}$ of the solution set Ω in an 'efficient' manner.

Before we present any specific algorithm for solving problem (9.1), we list certain desirable properties which we ideally expect the given algorithm to possess.

Definition 9.2.2 (Descent Property). *An algorithm for solving the unconstrained minimization problem (9.1) is said to have the descent property if the objective function value decreases as we go through the sequence $\{x^{(k)}\}$, i.e.*

$$f(x^{(k+1)}) < f(x^{(k)}) \text{ for all } k.$$

In other words, for the algorithm to possess the descent property, the objective function should decrease as we proceed.

Definition 9.2.3 (Quadratic Termination Property). *An algorithm for the unconstrained minimization problem (9.1) is said to have quadratic termination property (q.t.p) if the minimum of a positive definite quadratic form in n variables is reached in at most n iterations.*

The motivation for defining q.t.p stems from the fact that near a local min point, the function behaves like a strictly convex function(like a parabola in $\mathbf{R}$ or a positive definite quadratic form in $\mathbf{R}^n$) and therefore if the algorithm behaves well on such a function, it will 'hopefully' do well on the other functions as well.

Definition 9.2.4 (Globally Convergent). *An algorithm for the unconstrained minimization problem (9.1) is said to be globally convergent if starting from any point $x^{(0)} \in \mathbf{R}^n$, the sequence $\{x^{(k)}\}$ always converges to a point of the solution set Ω.*

The property of 'global convergence' guarantees that we can start the algorithm from any arbitrary point $x^{(0)} \in \mathbf{R}^n$. Thus no matter from which point we start, we are guaranteed to generate a sequence $\{x^{(k)}\}$ that converges to a point of the solution set. Here it must be noted that two different starting points will, in general, generate two different sequences of iterates but both converging to a point of solution set.

Definition 9.2.5 (Order of Convergence). *Let the sequence $\{x^{(k)}\}$ converge to a point $\bar{x}$ and let $x^{(k)} \neq \bar{x}$ for sufficiently large k. The quantity $\|x^{(k)} - \bar{x}\|$ is called the error of the k^{th} iterate $x^{(k)}$. Suppose that there exists p and $0 < a < \infty$ such that*

$$\lim_{k \to \infty} \frac{\|x^{(k+1)} - \bar{x}\|}{\|x^{(k)} - \bar{x}\|^p} = a \quad (0 < a < \infty)$$

then p is called the order of convergence of the sequence $\{x^{(k)}\}$. Thus $\|x^{(k+1)} - \bar{x}\| = a\|x^{(k)} - \bar{x}\|^p$ asymptotically.

If $p = 1$, the sequence $\{x^{(k)}\}$ is said to have *linear convergence rate* and for $p = 2$, it is said to have *quadratic convergence rate*. In case $p = 1$ but $a = 0$, then the sequence $\{x^{(k)}\}$ is said to have *super linear convergence rate*.

The order of convergence or convergence rate is an important concept as it tells us how the 'tail' of the sequence $\{x^{(k)}\}$ behaves. Larger values of p will imply faster convergence. Most of the algorithms generally do very well for the first few iterations but become very slow near the optimal solution. But if p is large then there will be significant improvement in the objective function value even near the actual solution.

We now discuss *line search methods* or *one dimensional search methods* which apart from being useful in themselves are used to determine the step size $\overline{\alpha_k}$ in the basic scheme

$$x^{(k+1)} = x^{(k)} + \overline{\alpha_k}\, d^{(k)} .$$

9.3 Line Search Methods for Unimodal Functions

These methods are used to locate min (or max) point of a *unimodal* function of one variable over a interval, i.e. we are given a unimodal function $f : [a, b] \longrightarrow \mathbf{R}$ and we wish to locate the minimizing point x_{min} such that

$$f(x_{min}) = \underset{a \leq x \leq b}{Min}\, f(x) . \tag{9.2}$$

Definition 9.3.1 (Unimodal Min Function). *The function $f : [a, b] \longrightarrow \mathbf{R}$ is said to be a unimodal (to be specific unimodal min) function if it has only one mode i.e. it has a single relative min, i.e. $\exists\ a \leq \alpha \leq b$ such that*
(i) f is strictly decreasing ($\downarrow$) in $[a, \alpha)$.
(ii) f is strictly increasing ($\uparrow$) in $[\alpha, b]$.

Similarly we define a *unimodal max function*. Note that a unimodal function may not be differentiable - in fact it may not even be continuous, as depicted in Fig 9.1.

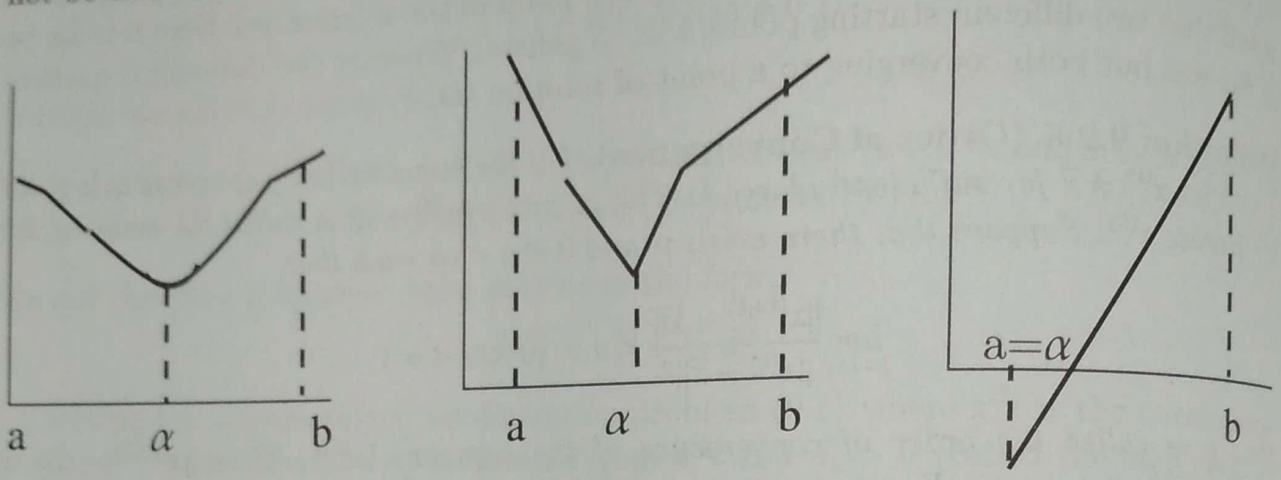


Fig. 9.1.

Basic Strategy

Let us choose two distinct points (say) x_1 and x_2 in $[a,b]$ i.e. $a \leq x_1 < x_2 \leq b$. Let x_{min} denote the point giving the minimum value of $f(x)$ over $[a,b]$. Since f is unimodal min function, it is clear that

(i) $f(x_1) < f(x_2) \Rightarrow x_{min} \in [a, x_2]$
(ii) $f(x_1) > f(x_2) \Rightarrow x_{min} \in [x_1, b]$
(iii) $f(x_1) = f(x_2) \Rightarrow x_{min} \in [x_1, x_2]$.

Let us agree to include case (iii) in case (i) (as case (iii) is not that likely). Then we observe that by evaluating the value of the function at two distinct points, the initial search length $(b-a)$(i.e. the interval $[a,\ b]$) has been reduced to (x_2-a) (i.e. the interval $[a,\ x_2]$) or $(b-x_1)$ (i.e. the interval $[x_1,\ b]$). The interval $[a,b]$ is called the initial *interval of uncertainty*. The basic question in line search methods of this type is to develop strategies for choosing points $x_1, x_2, ..., x_N$ such that $a \leq x_1 < x_2 < ... < x_{N-1} < x_N \leq b$ so that we can determine the smallest possible interval of uncertainty in which the minimizing point must lie. Note that we do not need explicit knowledge of the function f as long as it is possible to get its value at a specified point. Further we do not make any continuity/diffrentiability assumption on f. But then we have to assume that f is a unimodal min function.

There could be various search methods depending upon how the N trial points $x_1, x_2, ..., x_N$ have been chosen. It should be noted that because every function evaluation may be costly in time, we do not wish to evaluate the function f at too many points.

We shall be studying the following two search methods for the case of unimodal function. These are

(i) The Golden Section Rule.
(ii) The Fibonacci Search Method.

In discussing the aforesaid methods we shall make use of the following notations

$x_{L,k}$ = lower limit of the search interval at the k^{th} iteration (so $x_{L,1}$ = a)
$x_{U,k}$ = upper limit of the search interval at the k^{th} iteration (so $x_{U,1}$ = b)
$x_{p,k}$ = 1^{st} trial point at the k^{th} iteration
$x_{q,k}$ = 2^{nd} trial point at the k^{th} iteration
$E_{p,k} = f(x_{p,k})$ value of the function at the 1^{st} trial point
$E_{q,k} = f(x_{q,k})$ value of the function at the 2^{nd} trial point
$I_k = x_{U,k} - x_{L,k}$ = length of the search interval at the k^{th} iteration (so I_1 = (b - a))
I_k^L = length of the left part of the search interval I_k

I_k^R = length of the right part of the search interval I_k.

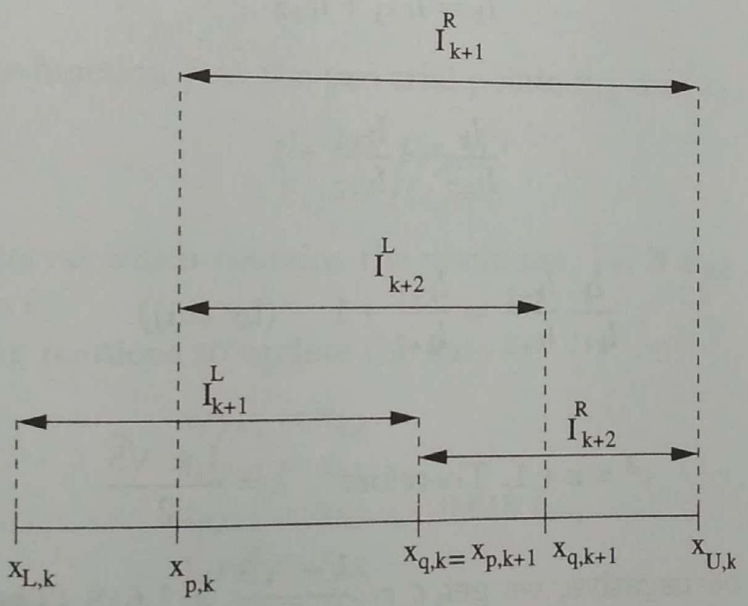


Fig. 9.2.

The Golden Section Rule

Here the two trial points $x_{p,k}$ and $x_{q,k}$ are chosen as per the following criteria
(i) $I_k^L = I_k^R$ for all k , i.e. no undue advantage is given to left or right part.
(ii) $x_{p,k+1} = x_{q,k}$ and $x_{q,k+1}$ is computed afresh for the interval I_{k+1}^R (or $x_{q,k+1} = x_{p,k}$ and $x_{p,k+1}$ is computed afresh for the interval I_{k+1}^L), i.e. only one new trial point is computed

at the k^{th} iteration, the other trial point is obtained from the previous iteration. In view of this we have

$$I_k = I_{k+1} + I_{k+2} \quad \text{for all } k.$$

(see Fig 9.2.)

(iii) $\dfrac{I_k}{I_{k+1}} = \dfrac{I_{k+1}}{I_{k+2}} = \text{c (constant) for all } k.$

This is called the *golden section criterion*. In general, a point w is said to divide the interval $[u, v]$ according to the golden section criterion if we have

$$\frac{\text{length of the whole interval}}{\text{length of the bigger interval}} = \frac{\text{length of the bigger interval}}{\text{length of the smaller interval}}$$

i.e. $$\frac{(v-u)}{(w-u)} = \frac{(w-u)}{(v-w)}.$$

Thus we wish to choose trial points $x_{p,k}$ and $x_{q,k}$ according to the above listed criterion (i), (ii) and (iii). From (ii) we have

$$I_k = I_{k+1} + I_{k+2}$$

i.e.

$$\frac{I_k}{I_{k+2}} = \frac{I_{k+1}}{I_{k+2}} + 1$$

i.e.

$$\frac{I_k}{I_{k+1}}\frac{I_{k+1}}{I_{k+2}} = \frac{I_{k+1}}{I_{k+2}} + 1 \quad \text{(by (iii))}$$

i.e.

$$c^2 = c + 1. \text{ Therefore } \quad c = \frac{1 \pm \sqrt{5}}{2}$$

Since c can not be negative, we get $c = \dfrac{1+\sqrt{5}}{2} = 1.618$ (The number $c = 1.618$ is called the *golden section ratio* and it has a long history - something to do with our aesthetic value). Note that $\dfrac{1}{1.618} = 0.618$.

Let us now imagine that the initial search interval is [0, 1]. Then due to the above arguments

$$\frac{1 - x_{p,1}}{x_{p,1}} = \frac{1}{1 - x_{p,1}} = 1.618.$$

Therefore
$$x_{p,1} = 0.382$$
$$x_{q,1} = 1 - 0.382 = 0.618 .$$

For a general interval $[a, b]$, we can then have

$$x_{p,1} = b - 0.618\,(b-a)$$
$$x_{q,1} = a + 0.618\,(b-a) .$$

Thus in general,

$$x_{p,k} = x_{U,k} - 0.618\, I_k$$
$$x_{q,k} = x_{L,k} + 0.618\, I_k .$$

The stepwise description of the method is now as follows

Step 1 Input data- $x_{L,1}$, $x_{U,1}$, ϵ, f (Here $\epsilon > 0$ is the tolerance to be prescribed by the user).

Step 2 Compute the first two trial points $x_{p,1}$ and $x_{q,1}$, where

$$x_{p,1} = x_{U,1} - 0.618\,(x_{U,1} - x_{L,1})$$
$$x_{q,1} = x_{L,1} + 0.618\,(x_{U,1} - x_{L,1}).$$

Set $k = 1$.

Step 3 Evaluate the function f at the two trial points $x_{p,k}$ and $x_{q,k}$. Let

$$E_{p,k} = f(x_{p,k})$$
$$E_{q,k} = f(x_{q,k}) .$$

Step 4 Test the interval which contains the minimum, i.e. if $E_{p,k} \le E_{q,k}$, go to Step 5 otherwise go to Step 6.

Step 5 Use following relations to update the data

$$x_{L,k+1} = x_{L,k}$$
$$x_{U,k+1} = x_{q,k}$$
$$x_{p,k+1} = x_{U,k+1} - 0.618\, I_{k+1}$$
$$x_{q,k+1} = x_{p,k}$$
$$E_{p,k+1} = f(x_{p,k+1})$$
$$E_{q,k+1} = f(x_{q,k+1}) = E_{p,k} .$$

Step 6 Use the following relations to update the data

$$x_{L,k+1} = x_{p,k}$$
$$x_{U,k+1} = x_{U,k}$$
$$x_{p,k+1} = x_{q,k}$$
$$x_{q,k+1} = x_{L,k} + 0.618\, I_{k+1}$$
$$E_{p,k+1} = f(x_{p,k+1}) = E_{q,k}$$
$$E_{q,k+1} = f(x_{q,k+1}) .$$

Step 7 Test for the end of optimization , i.e. if $I_k < \epsilon$, go to Step 8, otherwise set k = (k+1) and go to Step 4.
Step 8 Output $x_{L,n}$, $x_{U,n}$. Then $x_{min} \in (x_{L,n}, x_{U,n})$ and $E_{min} \leq Min(E_{p,n-1}, E_{q,n-1})$.

Remark 9.3.1 *It can be noted that for the golden section rule,* $\frac{I_n}{I_1} = \left(\frac{1}{1.618}\right)^{n-1} \approx (0.618)^{n-1}$. *Thus knowing* I_1 *(the length of the initial search interval) and* I_n *(the length of the final search interval as desired by the user) we can find out the number of iterations as well as the number of points at which the function is to be evaluated. Note that if we are going upto the* 7^{th} *iteration (i.e. getting* $x_{L,7}$ *and* $x_{U,7}$*) then we shall be having* $n = 7$ *functional evaluations, namely, at* $x_{p,1}$, $x_{q,1}$, *and one each at* 2^{nd}, 3^{rd}, 4^{th}, 5^{th} *and* 6^{th} *iteration. To stop at the* 7^{th} *iteration we shall be computing* $x_{p,6}$ *and* $x_{q,6}$ *and* $E_{min} \leq$ *Min* $(x_{p,6}, x_{q,6})$.

Thus N functional evaluations will mean stopping at the N^{th} *iteration and computing points upto* $x_{p,N-1}$ *and* $x_{q,N-1}$. *Also* $E_{min} \leq$ *Min* $(x_{p,N-1}, x_{q,N-1})$ *and there are only* $(N-1)$ *interval reductions.*

Example 9.3.1 *Find min* x^2 *over [-5, 15] by the golden section rule. Take* $\epsilon = 1.5$.

Solution Following the steps of the golden section rule we get

k	$x_{L,k}$	$x_{U,k}$	$x_{p,k}$	$x_{q,k}$	$E_{p,k}$	$E_{q,k}$	L/R
1	- 5.0	15.00	2.64	7.36	6.96	54.1	L
2	-5.0	7.36	-0.2	2.64	0.077	6.96	L
3	-5.0	2.64	-2.08	-0.27	4.33	0.077	R
4	-2.08	2.64	-0.2	0.84	0.077	0.71	L
5	-2.08	0.84	-0.96	-0.27	0.92	0.077	R
6	-0.96	0.84	-0.27	0.15	0.077	0.023	R
7	-0.27	0.84					

Here $I_7 = 1.11 < 1.5(= \epsilon)$, so we stop. Thus $x_{min} \in$ [-0.27, 0.84] and $E_{min} \leq$ Min (0.077, 0.023) = 0.023.

Note that this example is just for the sake of illustration. Normally the function will not be so well behaved and ϵ (tolerance) will be taken much smaller e.g. $\epsilon = 0.001$ etc.

The Fibonacci Search Method

We continue our discussion with regard to problem (9.2) and present another approach which is based on the Fibonacci sequence.

Definition 9.3.2 (Fibonacci Sequence). *Let $F_0 = 1$, $F_1 = 1$ and $F_i = F_{i-1} + F_{i-2}$ $(i \geq 2)$. Then $\{F_n\}$ is called the sequence of Fibonacci numbers or in short, the Fibonacci sequence. Thus the Fibonacci sequence is $\{1, 1, 2, 3, 5, 8, 13, 21, \ldots\}$ where $F_0 = 1$, $F_1 = 1$, $F_2 = 2$, $F_3 = 3$, $F_4 = 5$, $F_5 = 8$, $F_6 = 8$, $F_7 = 21 \ldots$ and so on.*

Now we describe the Fibonacci search scheme. Here the first two conditions are exactly same as in the golden section rule, but the third condition changes. Thus we have

(i) $I_k^L = I_k^R$ for all k
(ii) $I_k = I_{k+1} + I_{k+2}$
(iii) $\dfrac{I_k}{I_{k+1}} = \dfrac{F_{n-k+1}}{F_{n-k}}$, where n denotes the number of iterations to be performed.

We now give a justification of the condition (iii) given above. If we wish to stop at the n^{th} iteration, then

$$\begin{aligned}
I_{n+1} &\simeq I_n && = 1\, I_n \\
I_n &= I_n && = 1\, I_n \\
I_{n-1} &= I_n + I_{n+1} && = 2\, I_n \\
I_{n-2} &= I_n + I_{n-1} && = 3\, I_n \\
I_{n-3} &= 5\, I_n \\
I_{n-4} &= 8\, I_n \\
&\vdots \\
I_1 &= F_n I_n.
\end{aligned}$$

Thus we can see at once that the role of the Fibonacci numbers, because $I_1/I_n = F_n$, or in general, $\dfrac{I_k}{I_{k+1}} = \dfrac{F_{n-k+1}}{F_{n-k}}$. Hence the only difference between the golden section rule and the Fibonacci search method is the requirement (iii). In the golden section rule $\dfrac{I_k}{I_{k+1}} = c = 1.618$ for all k and all n. Here, in the Fibonacci search method this ratio depends on both the current iteration k and also the total number of iterations n to be performed. Therefore instead of 0.618 (i.e. $\dfrac{1}{c}$) we shall be using the number $\dfrac{F_{n-k}}{F_{n-k+1}}$ for determining the points $x_{p,k}$ and $x_{q,k}$.

The stepwise description of the Fibonacci search method is as follows

Step 1 Input data: $x_{L,1}$, $x_{U,1}$, n, f.
Step 2 Compute Fibonacci numbers F_n and F_{n-1} and find

$$x_{p,1} = x_{U,1} - \frac{F_{n-1}}{F_n}(x_{U,1} - x_{L,1})$$
$$x_{q,1} = x_{L,1} + \frac{F_{n-1}}{F_n}(x_{U,1} - x_{L,1})$$

Set $k = 1$.

Step 3 Evaluate

$$E_{p,k} = f(x_{p,k})$$
$$E_{q,k} = f(x_{q,k})$$

and test the interval which contains the minimum. If $E_{p,k} \leq E_{q,k}$, go to Step 4, otherwise go to Step 5.

Step 4 Use the following relations to update the data

$$x_{L,k+1} = x_{L,k}$$
$$x_{U,k+1} = x_{q,k}$$
$$x_{p,k+1} = x_{U,k+1} - \frac{F_{n-k}}{F_{n-k+1}} I_{k+1}$$
$$x_{q,k+1} = x_{p,k}$$
$$E_{p,k+1} = f(x_{p,k+1})$$
$$E_{q,k+1} = f(x_{q,k+1}) = E_{p,k} \ .$$

Step 5 Use the following relations to update the data

$$x_{L,k+1} = x_{p,k}$$
$$x_{U,k+1} = x_{U,k}$$
$$x_{p,k+1} = x_{q,k+1}$$
$$x_{q,k+1} = x_{L,k} + \frac{F_{n-k}}{F_{n-k+1}} I_{k+1}$$
$$E_{p,k+1} = f(x_{p,k+1}) = E_{q,k}$$
$$E_{q,k+1} = f(x_{q,k+1}) \ .$$

Step 6 If $k \leq (n-2)$, then set $k = k+1$ and go to step 3. If $k = (n-1)$ then go to Step 7.

Step 7 As $F_0 = F_1 = 1$, it can be seen that at the $(n-1)^{th}$ iteration, the two trial points $x_{p,n-1}$ and $x_{q,n-1}$ will come out to be the same , i.e. $x_{p,n-1} = x_{q,n-1}$. To make them distinct we have to add / subtract a fixed positive number ϵ (say 0.01) to one of these. For this we look at the $(n-2)^{th}$ iteration. If there (i.e. at the $(n-2)^{th}$ iteration) we are searching in the left then ϵ is subtracted from $x_{p,n-1}$ otherwise it is added to $x_{q,n-1}$. We then determine new $x_{p,n-1}$ and $x_{q,n-1}$ and hence $x_{L,n}$ and $x_{U,n}$. Then

$$x_{min} \in (x_{L,n}, x_{U,n})$$
and
$$f_{min} \leq \text{Min}(E_{p,n-1}, E_{q,n-1})$$

Remark 9.3.2 *As* $\frac{I_n}{I_1} = \frac{1}{F_n}$, *so given* I_1 *and* I_n, *the value of* n *can be computed in advance.*

Example 9.3.2 *Taking $n = 7$, find min x^2 over [-5,15] by the Fibonacci search method.*

Solution Following the steps of the Fibonacci search method we get

k	F_{n-k}/F_{n-k+1}	$x_{L,k}$	$x_{U,k}$	$x_{p,k}$	$x_{q,k}$	$E_{p,k}$	$E_{q,k}$	L/R
1	13/21	-5.0	15.00	2.64	7.38	6.88	54.6	L
2	8/13	-5.0	7.38	-0.24	2.62	0.058	6.88	L
3	5/8	-5.0	2.62	-2.14	-0.24	4.67	0.058	R
4	3/5	-2.14	2.62	-0.24	0.72	0.058	0.52	L
5	2/3	-2.14	0.72	-1.2	-0.24	1.44	0.058	R
6	1/2	- 1.2	0.72	-0.24	-0.24+0.01	0.058	0.053	R
7	1	-0.24	0.72					

From this table we observe that $x_{min} \in [-0.24, 0.72]$ and $f_{min} \leq 0.053$.

In the given example, $I_1 = 20$ and therefore if we take $\epsilon = 1.5$, then $I_n = 1.5$. This gives $\frac{I_1}{I_n} = \frac{20}{1.5} = 13.3$ which from the sequence of Fibonacci numbers determines $n = 7$ as $F_6 = 13$ and $F_7 = 21$. Also as explained in Step 7, the two trial points at the $(n-1)^{th}$ iteration, i.e. $x_{p,6}$ and $x_{q,6}$ both becomes equal to 0.24. Therefore we look at the $(n-2)^{th}$ iteration, i.e. the 5^{th} iteration in our example. As there we are seeking the right part, we add 0.01 to $x_{q,6}$ to get new $x_{q,6}$ and then continue as explained.

Relation Between the Fibonacci Search Method and the Golden Section Rule

For the constant c appearing in the golden section rule we have $c = 1 + 1/c = 1.618$. Also it can be proved that $F_n \simeq \frac{c^{n+1}}{\sqrt{5}}$ (for large n). We shall now like to compare these two methods for the same number of function evaluations.

Let R_F and R_{GS} denote the reduction factors for the Fibonacci search method and the golden section rule respectively. Then

$$R_F = \frac{I_n}{I_1} = \frac{1}{F_n} = \frac{\sqrt{5}}{c^{n+1}}$$

and

$$R_{GS} = \frac{I_n}{I_1} = \left(\frac{1}{c}\right)^{n-1} = \frac{1}{c^{n-1}}.$$

Therefore

$$\frac{R_{GS}}{R_F} = \frac{1}{c^{n-1}} \times \frac{c^{n+1}}{\sqrt{5}} = \frac{c^2}{\sqrt{5}} \simeq 1.17 .$$

Thus for the same number of function evaluations, the final search interval for the Fibonacci search method will be 17% smaller than the one obtained by the golden section rule .

9.4 The Steepest Descent Method

The steepest descent method is one of the oldest gradient based methods for solving an unconstrained optimization problem. This method is extremely simple to implement and therefore has been used widely in various applications. The only drawback of the steepest descent method is its slow convergence (as its order of convergence is one). But this has motivated researchers to develop more advanced algorithms by modifying the basic descent strategy of the steepest descent method so that these algorithms have superior convergence properties. We shall certainly discuss some of these algorithms in the subsequent sections.

Let us consider the unconstrained optimization problem (UMP)

$$\underset{x \in \mathbf{R}^n}{\text{Min}}\ f(x) \tag{9.3}$$

where f has continuous first order partial derivatives on $\mathbf{R}^n$. Then the basic scheme of the steepest descent method can be described as follows

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

where $d^{(k)} = -\nabla f(x^{(k)})/\|\nabla f(x^{(k)})\|$, is the direction of movement and the step size $\overline{\alpha_k} \geq 0$ is chosen such that

$$h(\overline{\alpha_k}) = \underset{\alpha_k \geq 0}{Min}\ h(\alpha_k).$$

Here the function $h(\alpha_k)$ is given by $h(\alpha_k) = f(x^{(k)} + \alpha_k\, d^{(k)})$, and we stop when $\|\nabla f(x^{(k)})\| < \epsilon$.

Thus a stepwise description of the steepest descent method could be

Step 1 Choose $x^{(0)} \in \mathbf{R}^n$ and tolerance $\epsilon > 0$. Set k = 0.

Step 2 Compute $\nabla f(x^{(k)})$ and $d^{(k)} = -\nabla f(x^{(k)})/\|\nabla f(x^{(k)})\|$.

Step 3 Evaluate $x^{(k+1)} = x^{(k)} + \overline{\alpha_k}\, d^{(k)}$ where $\overline{\alpha_k} > 0$ is chosen such that

$$h(\overline{\alpha_k}) = \underset{\alpha_k \geq 0}{Min}\ h(\alpha_k)\ ,$$

Here,

$$h(\alpha_k) = f(x^{(k)} + \alpha_k\, d^{(k)})\ .$$

The Steps 2 and 3 above are repeated till $\|\nabla f(x^{(k)})\| < \epsilon$. In that case $x^{(k)}$ becomes a point of the solution set and therefore, if f is a convex function then it becomes an optimal solution of the unconstrained minimization problem (9.3).

The natural question here is to justify the particular choice of the direction $d^{(k)}$ at the current point $x^{(k)}$, i.e. why should we choose $d^k = -\nabla f(x^{(k)})/\|\nabla f(x^{(k)})\|$? For this let us recall the definition of the directional derivative of f at the point x in the direction d as

$$\left.\frac{\partial f}{\partial d}\right|_x = \lim_{\alpha \to 0^+} \frac{f(x+\alpha d) - f(x)}{\alpha}.$$

and note that for the case when f has continuous first order partial derivatives, we have

$$\left.\frac{\partial f}{\partial d}\right|_x = \frac{d^T}{\|d\|} \nabla f(x).$$

Therefore for $d^{(k)} = -\dfrac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|}$, we get

$$\left.\frac{\partial f}{\partial d^{(k)}}\right|_{x=x^{(k)}} = -\frac{\|\nabla f(x^{(k)})\|^2}{\|\nabla f(x^{(k)})\|} = -\|\nabla f(x^{(k)})\| < 0.$$

As for $d^{(k)} = -\dfrac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|}$, $\left.\dfrac{\partial f}{\partial d^{(k)}}\right|_{x=x^{(k)}} < 0$, it makes sense to move in the direction $d^{(k)}$ if we wish to minimize the function $f(x)$. In fact the directional derivative of f at x in the direction d is least when $d = -\dfrac{\nabla f(x)}{\|\nabla f(x)\|}$ because for the given x the optimization problem

$$\begin{aligned} &\text{Min} && d^T \nabla f(x) \\ &\text{subject to} && \\ & && \|d\| \le 1. \end{aligned} \tag{9.4}$$

has the optimal solution $\bar{d} = -\dfrac{\nabla f(x)}{\|\nabla f(x)\|}$ with the optimal value $-\|\nabla f(x)\| < 0$.

Using any of the above arguments we must get convinced that if we are to minimize (locally) a function f from the current point $x^{(k)}$, then we must move in the direction of negative gradient and therefore we choose $d^{(k)}$ as the unit vector in that direction, i.e.

$$d^{(k)} = -\frac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|}.$$

Next point to understand is the procedure for choosing the step size $\bar{\alpha}_k$, where the direction of movement $d^{(k)}$ is known. For this we consider all points $x^{(k)} + \alpha_k d^{(k)}$, $\alpha_k > 0$,

i.e. all points on the ray originating at $x^{(k)}$ and in the direction $d^{(k)}$. Here it should be emphasized that as f is nonlinear, $\nabla f(x^{(k)})$ is a function of $x^{(k)}$, i.e. the function f is NOT going to decrease for all time to come, in the direction $d^{(k)}$, i.e. there must exist some $\overline{\alpha_k} > 0$ beyond which the function will not decrease in the direction $d^{(k)}$. A common sense argument suggests that the easiest (not necessarily the best) way to choose $\overline{\alpha_k}$ could be to consider all values $x^{(k)} + \alpha_k\, d^{(k)}$, i.e. consider the function $h(\alpha_k) = f(x^{(k)} + \alpha_k\, d^{(k)})$ and choose $\overline{\alpha_k} > 0$ for which $h(\alpha_k)$ is minimum. Here we should note that $h(\alpha_k)$ is a function of one variable, namely α_k, only and so this minimization can be done numerically as well. Therefore $\overline{\alpha_k} > 0$ is chosen such that

$$h(\overline{\alpha_k}) = \underset{\alpha_k > 0}{Min}\; h(\alpha_k)$$

as described in the basic scheme of the steepest descent method.

We should now go back and try to verify which of the so called 'desirable properties' the steepest descent method possesses. Without proving any of these we shall below state the given result (and this we shall follow for all algorithms discussed in this chapter) for the steepest descent method.

Result 9.4.1 *The steepest descent algorithm*
(i) has descent property
(ii) does not have quadratic termination property
(iii) is globally convergent and
(iv) has order of convergence $p = 1$.

Therefore if we are using the steepest descent method then we can start from any point $x^{(0)}$ and as we proceed, the objective function value will decrease, but the algorithm may take lot many iterations near the optimal solution. Also it may, in general, take more than n iterations to minimize a positive definite quadratic form of n variables.

We now illustrate the working of the steepest descent method for the example given below.

Example 9.4.1 *Use the steepest descent method to minimize* $f(x_1, x_2) = 3x_1^2 - 4x_1x_2 + 2x_2^2 + 4x_1 + 6$ *over* $(x_1, x_2) \in \mathbf{R}^2$.

Solution The given function is a convex function and so the steepest descent method will give a global optimal solution. Starting from $x^{(0)} = (0,0)^T$ and following Steps 1-3 described above, we get

k	$x^{(k)}$	$\nabla f(x^{(k)})$	$d^{(k)}$	$\overline{\alpha_k}$
1	$(0,0)^T$	$(4,0)^T$	$(-1,0)^T$	2/3
2	$(-2/3,0)^T$	$(0,8/3)^T$	$(0,-1)^T$	2/3
3	$(-2/3,-2/3)^T$	$(8/3,0)^T$	$(-1,0)^T$	1/6
4	$(-10/9,-2/3)^T$	$(0,16/9)^T$	$(0,-1)^T$	1/4
5	$(-38/27,-10/9)^T$	$(16/9,0)^T$	$(-1,0)^T$	1/6
⋮	⋮	⋮	⋮	⋮

Remark 9.4.1 *The function $f(x_1,x_2)$ of Example (9.4.1) is a positive definite quadratic form in two variables but its optimal solution has not been obtained in atmost two iterations. This illustrates that the method of steepest descent does not possess quadratic termination property.*

Remark 9.4.2 *Looking at the table for Example (9.4.1) we observe that the directions $d^{(k)}$ are repeated alternately $(-1,0)^T,(0,-1)^T,(-1,0)^T,(0,-1)^T$ etc. Is it a matter of coincidence or is it always going to happen? Well there is something more deeper here in the sense that any two consecutive directions $d^{(k)}$ and $d^{(k+1)}$ given by the steepest descent method are mutually orthogonal (see Theorem 9.4.1 below). Therefore in $\mathbf{R}^2$, if the first two directions are $d^{(1)}$ and $d^{(2)}$ then $d^{(3)}$ has to be $d^{(1)}$ and $d^{(4)}$ has to be $d^{(2)}$ so on. But this repetition may not be in $\mathbf{R}^3$ and higher dimensional spaces because there if $d^{(1)}$ and $d^{(2)}$ are orthogonal then we may have $d^{(3)}$, different from $d^{(1)}$, which is orthogonal to $d^{(2)}$. So the important thing is the orthogonality of consecutive directions and NOT their alternate repetition.*

Theorem 9.4.1 *Let $d^{(k)}$ and $d^{(k+1)}$ be two consecutive directions generated by the steepest descent method. Then*

$$< d^{(k+1)}, d^{(k)} >= 0$$

where $<,>$ denotes the standard inner product in $\mathbf{R}^n$.

Proof. Let $\overline{\alpha_k} > 0$ such that $h(\overline{\alpha_k}) = \underset{\alpha_k>0}{Min}\, h(\alpha_k)$. Then

$$\left.\frac{d\,h}{d\,\alpha_k}\right|_{\alpha=\overline{\alpha_k}} = 0,$$

i.e. $$\left.\frac{d\,[f(x^{(k)} + \alpha_k\, d^{(k)})]}{d\,\alpha_k}\right|_{\alpha=\overline{\alpha_k}} = 0$$

i.e. $$\left.\left(\nabla\, f(x^{(k)} + \alpha_k\, d^{(k)})\right)^T\, d^{(k)}\right|_{\alpha=\overline{\alpha_k}} = 0$$

i.e. $$\left.\left(\nabla\, f(x^{(k)} + \alpha_k\, d^{(k)}\right)^T \left(-\frac{\nabla f(x^{(k)})}{\|\nabla f(x^{(k)})\|}\right)\right|_{\alpha=\overline{\alpha_k}} = 0$$

i.e. $$\left(\nabla\, f(x^{(k)} + \overline{\alpha}_k\, d^{(k)})^T\right) \nabla\, f(x^{(k)}) = 0$$

i.e. $$< d^{(k+1)}, d^{(k)} > = 0\ .$$

□

Remark 9.4.3 *A geometrical interpretation of the above theorem could be that the vector* $-\nabla\, f(x^{(k)})$ *which is normal to the surface* $f(x) =$ *constant at* $x^{(k)}$, *is tangent to the surface at the point* $x^{(k+1)}$. *Therefore in* $\mathbf{R}^2$, *the movement of the steepest descent method will be as shown in Fig 9.3*

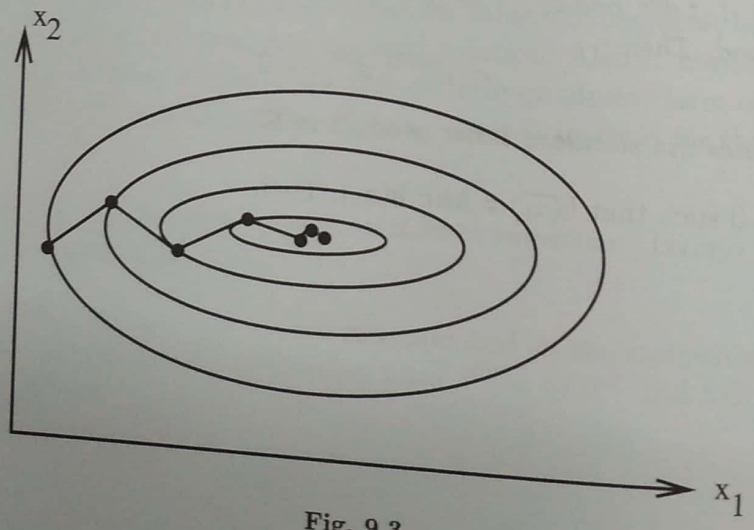


Fig. 9.3.

9.5 Newton's Method

Newton's method for finding real roots of the equation $g(y) = 0, y \in \mathbf{R}$ is well known. The basic scheme here is $y_{k+1} = y_k - g(y_k)/g'(y_k)$ where y_k is the current iterate or the current approximation.

It is natural here to be somewhat curious to know the connection between solving UMP's and the classical Newton's method for root finding. For this let us recollect that for solving the unconstrained minimization problem (9.3) we are aiming at finding a point $\bar{x} \in \mathbf{R}^n$ such that $\nabla f(\bar{x}) = 0$. Therefore the basic problem of root finding enters here very naturally except that rather than finding the roots of a single equation we have to find the roots of a system, namely $\nabla f(x) = 0$. Looking at the standard basic scheme of Newton's method for $g(y) = 0, y \in \mathbf{R}$, we immediately get the scheme

$$x^{(k+1)} = x^{(k)} - (H_f(x^{(k)}))^{-1} \nabla f(x^{(k)})$$

for finding a solution of the system $\nabla f(x) = 0$.

A more acceptable mathematical argument for the above scheme could be as follows. Let us approximate the given function f (note that $f : \mathbf{R}^n \longrightarrow \mathbf{R}$ is the given function in problem (9.3) which is to be optimized) in a neighborhood of the current approximate $x^{(k)}$ by the truncated Taylor series to get

$$f(x) \simeq f(x^{(k)}) + (x - x^{(k)})^T \nabla f(x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T H_f(x^{(k)})(x - x^{(k)}).$$

Therefore if we wish to minimize $f(x)$, it makes sense to minimize its quadratic approximation $q(x)$ where

$$q(x) = f(x^{(k)}) + (x - x^{(k)})^T \nabla f(x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T H_f(x^{(k)})(x - x^{(k)}).$$

Let this minimization be done exactly and hence

$$\nabla q(x) = 0,$$

i.e.

$$\nabla f(x^{(k)}) + \frac{1}{2}(x - x^{(k)})\, 2\, H_f(x^{(k)}) = 0$$

i.e.

$$(x - x^{(k)}) = -(H_f(x^{(k)}))^{-1} \nabla f(x^{(k)}),$$

which gives

$$x^{(k+1)} = x^{(k)} - (H_f(x^{(k)}))^{-1} \nabla f(x^{(k)}).$$

Since Newton's method for solving problem (9.3) essentially finds the roots of the system $\nabla f(x) = 0$, it follows that for minimizing a positive definite quadratic form of n variables, it will take exactly one iteration (imagine finding the roots of a linear

equation by Newton's method) because the system $\nabla f(x) = 0$ will be a linear system of equations. Therefore Newton's method for solving UMP's has quadratic termination property (in fact here for minimizing a positive definite quadratic form of n variables it will take exactly one iteration). Also it will have order of convergence $p = 2$ and it will have descent property, but it will not have the property of global convergence (because the standard Newton's method for root finding does not have this property). Therefore except for the case when we are minimizing a positive definite quadratic form (in the context of root finding it means solving a linear equation), we cannot start the method from an arbitrary point $x^{(0)} \in \mathbf{R}^n$.

Therefore if we are minimizing a positive definite quadratic form by Newton's method, then not only that we can start from any arbitrary point $x^{(0)} \in \mathbf{R}^n$, we also know that we will get the optimal solution in exactly one iteration, i.e. $x^{(1)}$ has to be the optimal solution $\bar{x}$. However if the function f in problem (9.3) is not a positive definite quadratic form then there are major problems with Newton's method. Apart from the fact that in this situation we cannot start from an arbitrary starting point $x^{(0)}$ ($x^{(0)}$ has to be 'close' to $\bar{x}$), there are serious issues with regard to the Hessian $H_f(x^{(k)})$. Why should $H_f(x^{(k)})$ be invertible at the point $x^{(k)}$ for every k ? It may be reasonable to assume that $H_f(\bar{x})$ is invertible in a neighborhood of $\bar{x}$ (because f has to behave like a 'parabola' or a positive definite quadratic form around a point which is a strict local min point) but assuming its invertibility for every $x^{(k)}$ does not make sense. However Newton's method has order of convergence $p = 2$ and this is very attractive because there will be significant improvement in the value of the objective function even when we are close to the actual minimizing point. (Recall that the order of convergence of the steepest descent method is $p = 1$ which makes the algorithm very slow near the actual optimal point).

In view of the above we should try to make certain modifications in Newton's method and get a method (say modified Newton's method) which has all the nice properties of Newton's method (descent property, quadratic termination property and order of convergence $p = 2$) and is also globally convergent (so that we can start from an arbitrary point $x^{(0)} \in \mathbf{R}^n$). Also it will be nice if the proposed method also takes care of the issues related with the existence of the inverse of the Hessian. We discuss modified Newton's method in the next section.

Example 9.5.1 *Use Newton's method to minimize* $f(x_1, x_2) = 8x_1^2 - 4x_1x_2 + 5x_2^2$, $(x_1, x_2) \in \mathbf{R}^2$.

Solution As the function f is a positive definite quadratic form in two variables we know for certain that we can start from any arbitrary point $x^{(0)} \in \mathbf{R}^2$ and use Newton's method to get $x^{(1)}$, then $x^{(1)}$ has to be the minimizing point.

To be specific, let $x^{(0)} = (5, 2)^T$. Then

$$\nabla f(x^{(0)}) = \begin{pmatrix} 16x_1 - 4x_2 \\ -4x_1 + 10x_2 \end{pmatrix}\Bigg|_{(x_1=5, x_2=2)}$$
$$= \begin{pmatrix} 72 & 0 \end{pmatrix},$$
$$H_f(x^{(0)}) = \begin{pmatrix} 16 & -4 \\ -4 & 10 \end{pmatrix},$$

and

$$(H_f(x^{(0)}))^{-1} = \frac{1}{144}\begin{pmatrix} 10 & 4 \\ 4 & 16 \end{pmatrix}.$$

Then

$$x^{(1)} = x^{(0)} - (H_f(x^{(0)}))^{-1}\nabla f(x^{(0)})$$
$$= \begin{pmatrix} 5 \\ 2 \end{pmatrix} - \frac{1}{444}\begin{pmatrix} 10 & 4 \\ 4 & 16 \end{pmatrix}\begin{pmatrix} 72 \\ 0 \end{pmatrix}$$
$$= \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

giving $\bar{x}_1 = 0, \bar{x}_2 = 0$ as the minimizing point.

9.6 Modified Newton's Method

While discussing Newton's method in the last section we noted certain limitations as $H_f(x^{(k)})$ may not be invertible at the points $x^{(k)}$. There is something more to it - namely, even if we could guarantee the existence of $(H_f(x^{(k)}))^{-1}$, it is not necessary that $-(H_f(x^{(k)}))^{-1}\nabla f(x^{(k)})$ will be the direction of descent unless we could also guarantee that $H_f(x^{(k)})$ is positive definite at $x^{(k)}$. For this it is enough to check that $-(M_k(x^{(k)}))^{-1}\nabla f(x^{(k)})$ is always a direction of descent for any positive definite matrix M_k. Another difficulty with Newton's method has been its lack of global convergence property. Keeping these things in mind, the following modification to Newton's method is suggested

$$x^{(k+1)} = x^{(k)} - \overline{\alpha_k}\, M_k\, (\nabla f(x^{(k)})) \tag{9.5}$$

where M_k is an appropriate positive definite matrix (obtained from $H_f(x^{(k)})$ as explained here) and $\overline{\alpha_k} > 0$ is the step size which is chosen as in the steepest descent method, i.e. $\overline{\alpha_k} > 0$ is chosen such that

$$h(\overline{\alpha_k}) = \underset{\alpha_k > 0}{Min}\, h(\alpha_k),$$

where

$$h(\alpha_k) = f(x^{(k)} + \alpha_k\, d^{(k)}),\ d^{(k)} = -M_k(\nabla f(x^{(k)})).$$

Now comes the main question. How should we choose M_k given the matrix $H_f(x^{(k)})$? Here we should remember that $H_f(x^{(k)})$ may not be invertible at $x^{(k)}$ and if at some point

$x^{(k)}$ it is invertible it is not necessary that its inverse is positive definite. But $H_f(x^{(k)})$ is certainly real symmetric and hence all its eigen values are real. What we shall do now is to add a suitable matrix of the form $\epsilon_k\, I$ $(\epsilon_k > 0)$ and take $M_k = (F_k)^{-1}$ where $F_k = (\epsilon_k\, I + H_f(x^{(k)}))$. Here $\epsilon_k > 0$ is to be chosen so that all eigenvalues of F_k become strictly positive and therefore F_k and M_k become positive definite. For this, given the point $x^{(k)}$, we fix a constant $\delta > 0$ and calculate all eigenvalues of $H_f(x^{(k)})$. Let ϵ_k be the smallest non negative constant for which all eigenvalues of the matrix $\epsilon_k\, I + H_f(x^{(k)})$ are greater than or equal to δ. Therefore, once $\epsilon > 0$ has been chosen in this manner, we take $F_k = (\epsilon_k\, I + H_f(x^{(k)}))$ and $M_k = (\epsilon_k\, I + H_f(x^{(k)}))^{-1}$.

It can be shown that with the above modification the method described above (called modified Newton's method) has all the nice properties, namely it has the descent property, it has quadratic termination property, it has property of global convergence and its order of convergence p is 2. However it is still not practical because to get M_k we need to compute all eigenvalues of $H_f(x^{(k)})$.

Therefore we now have two basic gradient based methods, namely the steepest descent method and the modified Newton's method, for solving unconstrained minimization problems. Whereas the steepest descent method is simple to implement, it is not very good from the convergence point of view, as its order of convergence p is one. The other method, namely the modified Newton's method has all the nice properties, including the global convergence and order two convergence, it is not of much use because of the effort involved in evaluating F_k and hence M_k. So the best option seems to be looking for those algorithms for UMP's which are somewhere in the middle of the spectrum, i.e. these are simpler to implement (unlike the modified Newton's method) and have better order of convergence (unlike the steepest descent method). There are a whole class of such methods, namely *conjugate direction methods* and *quasi Newton methods*. We shall discuss some of these in the next two sections.

9.7 The Conjugate Gradient Method

Here we present certain basic principles of conjugate direction methods for solving UMP's and discuss the conjugate gradient method in detail. As mentioned in the previous section, these methods are better than the steepest descent method (in terms of order of convergence) and are simpler to implement than the modified Newton's method.

Definition 9.7.1 (Conjugate Directions). *Let Q ne an $(n \times n)$ positive definite matrix. Any two non-zero vectors (directions) $d^{(1)}$, $d^{(2)} \in \mathbf{R}^n$ are said to be conjugate vectors or conjugate directions with respect to Q, if $(d^{(1)})^T\, Q\, d^{(2)} = 0$.*

Here we note that for $Q = I$, conjugacy reduces to the usual concept of orthogonality. Therefore if $d^{(1)}$ and $d^{(2)}$ are conjugate with respect to Q, sometimes we also call them Q - orthogonal.

$x^{(k)}$ it is invertible it is not necessary that its inverse is positive definite. But $H_f(x^{(k)})$ is certainly real symmetric and hence all its eigen values are real. What we shall do now is to add a suitable matrix of the form $\epsilon_k\, I$ $(\epsilon_k > 0)$ and take $M_k = (F_k)^{-1}$ where $F_k = (\epsilon_k\, I + H_f(x^{(k)}))$. Here $\epsilon_k > 0$ is to be chosen so that all eigenvalues of F_k become strictly positive and therefore F_k and M_k become positive definite. For this, given the point $x^{(k)}$, we fix a constant $\delta > 0$ and calculate all eigenvalues of $H_f(x^{(k)})$. Let ϵ_k be the smallest non negative constant for which all eigenvalues of the matrix $\epsilon_k\, I + H_f(x^{(k)})$ are greater than or equal to δ. Therefore, once $\epsilon > 0$ has been chosen in this manner, we take $F_k = (\epsilon_k\, I + H_f(x^{(k)}))$ and $M_k = (\epsilon_k\, I + H_f(x^{(k)}))^{-1}$.

It can be shown that with the above modification the method described above (called modified Newton's method) has all the nice properties, namely it has the descent property, it has quadratic termination property, it has property of global convergence and its order of convergence p is 2. However it is still not practical because to get M_k we need to compute all eigenvalues of $H_f(x^{(k)})$.

Therefore we now have two basic gradient based methods, namely the steepest descent method and the modified Newton's method, for solving unconstrained minimization problems. Whereas the steepest descent method is simple to implement, it is not very good from the convergence point of view, as its order of convergence p is one. The other method, namely the modified Newton's method has all the nice properties, including the global convergence and order two convergence, it is not of much use because of the effort involved in evaluating F_k and hence M_k. So the best option seems to be looking for those algorithms for UMP's which are somewhere in the middle of the spectrum, i.e. these are simpler to implement (unlike the modified Newton's method) and have better order of convergence (unlike the steepest descent method). There are a whole class of such methods, namely *conjugate direction methods* and *quasi Newton methods*. We shall discuss some of these in the next two sections.

9.7 The Conjugate Gradient Method

Here we present certain basic principles of conjugate direction methods for solving UMP's and discuss the conjugate gradient method in detail. As mentioned in the previous section, these methods are better than the steepest descent method (in terms of order of convergence) and are simpler to implement than the modified Newton's method.

Definition 9.7.1 (Conjugate Directions). *Let Q ne an $(n \times n)$ positive definite matrix. Any two non-zero vectors (directions) $d^{(1)}$, $d^{(2)} \in \mathbf{R}^n$ are said to be conjugate vectors or conjugate directions with respect to Q, if $(d^{(1)})^T\, Q\, d^{(2)} = 0$.*

Here we note that for $Q = I$, conjugacy reduces to the usual concept of orthogonality. Therefore if $d^{(1)}$ and $d^{(2)}$ are conjugate with respect to Q, sometimes we also call them Q - orthogonal.

The above definition is extended to more than two vectors in a natural manner, i.e. a set $\{d^{(0)}, \ldots, d^{(k)}\}$ of $(k+1)$ vectors in $\mathbf{R}^n$ is said to be conjugate if every two of them are so, i.e. $(d^{(i)})^T\ Q\ d^{(j)} = 0\ (i \neq j)$. Now onwards we shall not write 'conjugate with respect to Q' but rather just write 'conjugate' in case there is no confusion.

Result 9.7.1 *Let $\{d^{(0)}, d^{(1)}, \ldots, d^{(k)}\}$ be a set of $(k+1)$ non-zero vectors which are conjugate with respect to a given positive definite matrix Q. Then the vectors $d^{(0)}, d^{(1)}, \ldots, d^{(k)}$ are linearly independent.*

Proof. To prove the above result, we have to show that $\alpha_0\ d^{(0)} + \alpha_1\ d^{(1)} + \ldots + \alpha_k\ d^{(k)} = 0$ implies that each $\alpha_i = 0$. For this consider the equation

$$\alpha_0\ d^{(0)} + \alpha_1\ d^{(1)} + \ldots + \alpha_k\ d^{(k)} = 0$$

and pre-multiply both sides by $(d^{(i)})^T\ Q$. Then by the definition of conjugacy, we obtain

$$\alpha_i\ ((d^{(i)})^T\ Q\ d^{(i)}) = 0,$$

which gives $\alpha_i = 0$ as the matrix Q is positive definite. □

Now to understand the basic principles of conjugate direction methods, we first consider the quadratic case, i.e. the problem

$$\underset{x \in \mathbf{R}^n}{\text{Min}} \quad \frac{1}{2}\ x^T Q\ x - b^T x \tag{9.6}$$

where Q is an $(n \times n)$ symmetric positive definite matrix. As Q is positive definite the objective function of problem (9.6) is strictly convex and therefore problem has unique minimizing point $\bar{x} \in \mathbf{R}^n$.

Further the KKT conditions for problem (9.6) give $\nabla\left(\frac{1}{2}\ x^T Q x - b^T x\right) = 0$, i.e. $Qx = b$, which implies $\bar{x} = Q^{-1}b$ (note that as Q is positive definite, Q^{-1} exists).

The above discussion shows that finding the unique minimizing point $\bar{x}$ of problem (9.6) is equivalent to finding the unique solution of the system of equations $Qx = b$, namely $\bar{x} = Q^{-1}b$. Hence there is no theoretical difficulty as Q^{-1} can always be computed and so $\bar{x}$ can always be determined. The main purpose of introducing conjugate directions is to obtain this unique $\bar{x}$ without really finding Q^{-1} explicitly. This is something we always attempt in numerical optimization, i.e. we do not compute the inverses explicitly. We may recall here that the idea of pivoting used in the simplex algorithm was introduced so that we do not compute the inverse of the basis matrix explicitly.

In the following result, we demonstrate that if we can get hold of a set of n non-zero vectors which are Q - conjugate, then the desired $\bar{x}$ can be obtained easily and this will not need the computation of Q^{-1} explicitly.

Result 9.7.2 *Let $\{d^{(0)}, d^{(1)}, \ldots, d^{(n-1)}\}$ be a set of n non-zero vectors in $\mathbf{R}^n$ which are conjugate with respect to Q. Then $\bar{x}$, which is the unique solution to the system $Qx = b$ or equivalently, the unique minimizing point of problem (9.6), is given by*

$$\bar{x} = \sum_{k=0}^{n-1} \left(\frac{(d^{(k)})^T b}{(d^{(k)})^T Q d^{(k)}} \right) d^{(k)}. \tag{9.7}$$

Proof. Using Result 9.7.1, we note that the vectors $d^{(0)}, d^{(1)}, \ldots, d^{(n-1)}$ are linearly independent. As these vectors are exactly n in number, they form a basis of $\mathbf{R}^n$. Therefore there exist scalars $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}$ such that

$$\bar{x} = \alpha_0\, d^{(0)} + \alpha_1\, d^{(1)} + \ldots + \alpha_{n-1}\, d^{(n-1)}.$$

If we now pre-multiply the above equation by $(d^{(k)})^T Q$ and use the definition of conjugacy, we get

$$\alpha_k = \frac{(d^{(k)})^T Q \bar{x}}{(d^k)^T Q d^{(k)}}.$$

But $\bar{x}$ is the unique solution of $Q\,x = b$ i.e. $Q\,\bar{x} = b$ and hence

$$\alpha_k = \frac{(d^{(k)})^T b}{(d^{(k)})^T Q d^{(k)}}$$

which on substitution in the equation (9.7) gives the result. □

The above result can also be visualized as the output of an iterative process (see Theorem 9.7.1), which becomes very handy in describing the conjugate gradient method.

Theorem 9.7.1 (Conjugate Direction Theorem). *Let $\{d^{(0)}, d^{(1)}, \ldots, d^{(n-1)}\}$ be a set of n non-zero vectors in $\mathbf{R}^n$ which are conjugate with respect to Q. For any $x^{(0)} \in \mathbf{R}^n$, the sequence $\{x^{(k)}\}$ generated according to*

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \overline{\alpha_k}\, d^{(k)}, \\ \overline{\alpha_k} &= -\frac{(g^{(k)})^T\, d^{(k)}}{(d^{(k)})^T\, Q\, d^{(k)}}, \\ g^{(k)} &= Qx^{(k)} - b, \end{aligned}$$

converges to the unique solution $\bar{x}$ of the system $Q\,x = b$ exactly after n steps, i.e. $x^{(n)} = \bar{x}$.

Proof. Since $\{d^{(0)}, d^{(1)}, \ldots, d^{(n-1)}\}$ are Q - conjugate, they (as before) form a basis of $\mathbf{R}^n$, and so there exist scalars $\lambda_0, \lambda_1, \ldots, \lambda_{n-1}$ such that

$$\bar{x} - x^{(0)} = \lambda_0\, d^{(0)} + \lambda_1\, d^{(1)} + \ldots + \lambda_{n-1}\, d^{(n-1)}. \tag{9.8}$$

Now pre-multiplying the above equation by $(d^{(k)})^T Q$, we get

$$\lambda_k = \frac{(d^{(k)})^T Q(\bar{x} - x^{(0)})}{(d^{(k)})^T Q d^{(k)}} . \tag{9.9}$$

Also by following the iterative scheme as given in the hypothesis of the theorem, we get

$$\begin{aligned} x^{(1)} - x^{(0)} &= \alpha_0 d^{(0)} \\ x^{(2)} - x^{(0)} &= x^{(1)} + \alpha_1 d^{(1)} - x^{(0)} = \alpha_0\, d^{(0)} + \alpha_1\, d^{(1)} \\ &\vdots \end{aligned}$$

$$x^{(k)} - x^{(0)} = \alpha_0\, d^{(0)} + \alpha_1\, d^{(1)} + \ldots + \alpha_{k-1}\, d^{(k-1)} . \tag{9.10}$$

Again, the pre-multiplication of both sides of (9.10) by $(d^{(k)})^T Q$ gives

$$(d^{(k)})^T Q(x^{(k)} - x^{(0)}) = 0. \tag{9.11}$$

Therefore

$$\lambda_k = \frac{(d^{(k)})^T Q(\bar{x} - x^{(k)} + x^{(k)} - x^{(0)})}{(d^{(k)})^T Q d^{(k)}} \tag{9.12}$$

i.e,

$$\begin{aligned} \lambda_k &= \frac{(d^{(k)})^T Q(\bar{x} - x^{(k)})}{(d^{(k)})^T Q d^{(k)}} \text{ (by (9.11))} \\ &= \frac{(d^{(k)})^T (Q\bar{x} - Qx^{(k)})}{(d^{(k)})^T Q d^{(k)}} \\ &= \frac{(d^{(k)})^T (b - Qx^{(k)})}{(d^{(k)})^T Q d^{(k)}} \\ &= \frac{-(d^{(k)})^T g^{(k)}}{(d^{(k)})^T Q d^{(k)}} \\ &= \frac{-(g^{(k)})^T d^{(k)}}{(d^{(k)})^T Q d^{(k)}} \\ &= \alpha_k. \end{aligned}$$

Now by the iterative scheme

$$x^{(n)} - x^{(0)} = \alpha_0\, d^{(0)} + \alpha_1\, d^{(1)} + \ldots + \alpha_{n-1}\, d^{(n-1)}\,. \tag{9.13}$$

Also as given by (9.8),

$$\bar{x} - x^{(0)} = \lambda_0\, d^{(0)} + \lambda_1\, d^{(1)} + \ldots + \lambda_{n-1}\, d^{(n-1)}\,. \tag{9.14}$$

But as shown above, $\lambda_k = \alpha_k$ for all $k = 0, 1, \ldots, n-1$, and hence from (9.13)-(9.14)

$$x^{(n)} - x^{(0)} = \bar{x} - x^{(0)}$$

i.e.

$$x^{(n)} = \bar{x}.$$

In view of Result 9.7.1 and Theorem 9.7.1 we conclude that to solve problem (9.6) we need to obtain n non-zero conjugate directions $d^{(0)}, d^{(1)}, \ldots, d^{(n-1)}$. But how to determine these directions is the main question now. The applicability of conjugate direction methods will essentially depend on how simple or difficult is the method of finding these conjugate directions. In the following we present one such conjugate direction method where the conjugate directions are determined by using the gradient of the function f. This method therefore, is appropriately called *conjugate gradient method.*

Conjugate Gradient Method for the Quadratic Case

Let us again consider the unconstrained optimization problem (9.6), i.e.

$$\underset{x \in \mathbf{R}^n}{Min} \quad \frac{1}{2}\, x^T Q x - b^T x$$

where $b \in \mathbf{R}^n$ and Q is an $(n \times n)$ positive definite matrix. Let us also recall that solving the above problem is equivalent to finding the unique solution $\bar{x}$ of the system $Qx = b$. We first describe the conjugate gradient method and then later justify various steps involved therein.

Step 1 Choose $x^{(0)} \in \mathbf{R}^n$ arbitrary. Define $d^{(0)} = -g^{(0)} = b - Qx^{(0)}$. Set $k = 0$.

Step 2 Use the scheme

$$\begin{aligned}
x^{(k+1)} &= x^{(k)} + \overline{\alpha_k} d^{(k)} \\
\overline{\alpha_k} &= \frac{-(g^{(k)})^T d^{(k)}}{(d^{(k)})^T Q d^{(k)}} \\
d^{(k+1)} &= -g^{(k+1)} + \beta_k d^{(k)} \\
\beta_k &= \frac{(g^{(k+1)})^T Q d^{(k)}}{(d^{(k)})^T Q d^{(k)}} \\
g^{(k)} &= Qx^{(k)} - b\,.
\end{aligned}$$

Step 3 Continue till we get $x^{(n)}$. Then $x^{(n)} = \bar{x}$ and hence stop.

In the above we note that the first step is the same as in the steepest descent method. But there after, each successive step moves in a direction that is a linear combination of the current gradient and the preceding direction vector. Here the formula for the computation of the scalar β_k has been derived so as to guarantee that $d^{(k+1)}$ is conjugate with respect to all previous directions, i.e. $(d^{(k+1)})^T\ Q\ d^{(i)} = 0$, for $i = 0,1,\ldots,k$. The main point to note here is that we do not need all directions $d^{(0)}, d^{(1)}, \ldots, d^{(k)}$ at one go. But, rather we start from $d^{(0)}$ and generate subsequent conjugate directions as we proceed with the algorithm. As soon as we determine $d^{(n-1)}$, the point $x^{(n)}$ is known and that is precisely the point $\bar{x}$. The mathematical justification of conjugate gradient follows from the conjugate direction theorem, i.e. Theorem 9.7.1.

Example 9.7.1 *Use the conjugate gradient method to minimize* $f(x_1, x_2) = 3x_1^2 - 4x_1x_2 + 2x_2^2 + 4x_1 + 6,\ (x_1, x_2) \in \mathbf{R}^2$.

Solution To use the conjugate gradient method we need to express the given quadratic function in the form $f(x) = \frac{1}{2}\, x^T Q x - b^T x$. It is simple to check here that $Q = \begin{pmatrix} 6 & -4 \\ -4 & 4 \end{pmatrix}$, $b = \begin{pmatrix} -4 \\ 0 \end{pmatrix}$ and $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.

Further we also check that Q is positive definite so the given problem is exactly in the form of the unconstrained optimization problem (9.6). Also the objective function is a positive definite quadratic form in two variables, and therefore by the conjugate direction theorem we know that starting with any arbitrary point $x^{(0)} \in \mathbf{R}^2$ and following the steps of conjugate gradient method, once we compute $x^{(2)}$ then $x^{(2)}$ has to be the unique minimizing point $\bar{x}$. For the sake of illustration let $x^{(0)} = (0,0)^T$.

Step 1 For $x^{(0)} = (0,0)^T, g^{(0)} = Qx^{(0)} - b = \begin{pmatrix} 6 & -4 \\ -4 & 4 \end{pmatrix}\begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} -4 \\ 0 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$.

Therefore, $d^{(0)} = -g^{(0)} = (-4,0)^T$.

Step 2 Next $x^{(1)} = x^{(0)} + \overline{\alpha_0} d^{(0)}$ where

$$\overline{\alpha_0} = \frac{-(g^{(0)})^T d^{(0)}}{(d^{(0)})^T Q d^{(0)}} = \frac{-(4,0)\begin{pmatrix} 4 \\ 0 \end{pmatrix}}{(-4,0)\begin{pmatrix} 6 & -4 \\ -4 & 4 \end{pmatrix}\begin{pmatrix} -4 \\ 0 \end{pmatrix}} = 1/6$$

Therefore,

$$x^{(1)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + 1/6 \begin{pmatrix} -4 \\ 0 \end{pmatrix} = \begin{pmatrix} -2/3 \\ 0 \end{pmatrix}$$

Step 3 Now,

$$g^{(1)} = Qx^{(1)} - b = \begin{pmatrix} 6 & -4 \\ -4 & 4 \end{pmatrix}\begin{pmatrix} -2/3 \\ 0 \end{pmatrix} - \begin{pmatrix} -4 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 8/3 \end{pmatrix},$$

$$\beta_0 = \frac{(g^{(1)})^T Qd^{(0)}}{(d^{(0)})^T Qg^{(0)}} = \frac{(0, 8/3)\begin{pmatrix} 6 & -4 \\ -4 & 4 \end{pmatrix}\begin{pmatrix} -4 \\ 0 \end{pmatrix}}{96} = 4/9,$$

and $d^{(1)} = -g^{(1)} + \beta_0 d^{(0)} = \begin{pmatrix} 0 \\ -8/3 \end{pmatrix} + \frac{4}{9}\begin{pmatrix} -4 \\ 0 \end{pmatrix} = \begin{pmatrix} -16/9 \\ -8/3 \end{pmatrix}$.

Step 4 Now we obtain $x^{(2)}$ as
$x^{(2)} = x^{(1)} + \overline{\alpha_1} d^{(1)}$. But

$$\overline{\alpha_1} = \frac{-(g^{(1)})^T d^{(1)}}{(d^{(1)})^T Q d^{(1)}} = 3/4,$$

Therefore,

$$x^{(2)} = \begin{pmatrix} -2/3 \\ 0 \end{pmatrix} + \frac{3}{4}\begin{pmatrix} -16/9 \\ -8/3 \end{pmatrix}$$
$$= \begin{pmatrix} -2 \\ -2 \end{pmatrix}$$
$$= \overline{x},$$

i.e. the minimizing point of $f(x_1, x_2)$ is $\overline{x}_1 = -2$, $\overline{x}_2 = -2$.

Remark 9.7.1 *Here we can verify that* $\nabla f(\overline{x})$, *i.e.* $Q\overline{x} - b = 0$ *as it should be. Also* $(d^{(1)})^T Q\, d^{(0)} = 0$. *Again this has to be true because the basic argument of conjugate gradient method is to generate directions* $d^{(k)}$ *such that* $d^{(0)}, d^{(1)}, \ldots, d^{(n-1)}$ *are conjugate with respect to* Q.

Result 9.7.3 *For the conjugate gradient method, following are true*

(i) $(g^{(k+1)})^T d^{(k)} = 0$

(ii) $\overline{\alpha_k} = \dfrac{(g^{(k)})^T g^{(k)}}{(d^{(k)})^T Q\, d^{(k)}}$

(iii) $\beta_k = \dfrac{(g^{(k+1)})^T g^{(k+1)}}{(g^{(k)})^T g^{(k)}}$

(iv) $(d^{(k)})^T Q\, d^{(i)} = 0$, $(i = 0, 1, \ldots, k-1)$.

Here $g^{(k)}$ *is the gradient of the objective function at* $x^{(k)}$ *i.e.* $g^{(k)} = Q\, x^{(k)} - b$.

Proof. We shall prove (i), (ii) and (iii) only. As far as (iv) is concerned, from the definition of β_k, it is simple to prove that $(d^{(k+1)})^T Q d^{(k)} = 0$ and then to prove that $(d^{(k+1)})^T Q d^{(i)} = 0$ for all $i = 0, 1, \ldots, k-1$, we have to use induction. For full proof of this, we shall refer to Luenberger [106].

CHAPTER 8

Calculating Derivatives

Most algorithms for nonlinear optimization and nonlinear equations require knowledge of derivatives. Sometimes the derivatives are easy to calculate by hand, and it is reasonable to expect the user to provide code to compute them. In other cases, the functions are too complicated, so we look for ways to calculate or approximate the derivatives automatically. A number of interesting approaches are available, of which the most important are probably the following.

Finite Differencing. This technique has its roots in Taylor's theorem (see Chapter 2). By observing the change in function values in response to small perturbations of the unknowns

near a given point x, we can estimate the response to *infintesimal* perturbations, that is, the derivatives. For instance, the partial derivative of a smooth function $f : \mathbb{R}^n \to \mathbb{R}$ with respect to the ith variable x_i can be approximated by the central-difference formula

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon},$$

where ϵ is a small positive scalar and e_i is the ith unit vector, that is, the vector whose elements are all 0 except for a 1 in the ith position.

Automatic Differentiation. This technique takes the view that the computer code for evaluating the function can be broken down into a composition of elementary arithmetic operations, to which the chain rule (one of the basic rules of calculus) can be applied. Some software tools for automatic differentiation (such as ADIFOR [25]) produce new code that calculates both function and derivative values. Other tools (such as ADOL-C [154]) keep a record of the elementary computations that take place while the function evaluation code for a given point x is executing on the computer. This information is processed to produce the derivatives at the same point x.

Symbolic Differentiation. In this technique, the algebraic specification for the function f is manipulated by symbolic manipulation tools to produce new algebraic expressions for each component of the gradient. Commonly used symbolic manipulation tools can be found in the packages Mathematica [311], Maple [304], and Macsyma [197].

In this chapter we discuss the first two approaches: finite differencing and automatic differentiation.

The usefulness of derivatives is not restricted to *algorithms* for optimization. Modelers in areas such as design optimization and economics are often interested in performing post-optimal *sensitivity analysis*, in which they determine the sensitivity of the optimum to small perturbations in the parameter or constraint values. Derivatives are also important in other areas such as nonlinear differential equations and simulation.

8.1 FINITE-DIFFERENCE DERIVATIVE APPROXIMATIONS

Finite differencing is an approach to the calculation of approximate derivatives whose motivation (like that of so many algorithms in optimization) comes from Taylor's theorem. Many software packages perform automatic calculation of finite differences whenever the user is unable or unwilling to supply code to calculate exact derivatives. Although they yield only approximate values for the derivatives, the results are adequate in many situations.

By definition, derivatives are a measure of the sensitivity of the function to infinitesimal changes in the values of the variables. Our approach in this section is to make small, *finite* perturbations in the values of x and examine the resulting *differences* in the function values.

By taking ratios of the function difference to variable difference, we obtain approximations to the derivatives.

APPROXIMATING THE GRADIENT

An approximation to the gradient vector $\nabla f(x)$ can be obtained by evaluating the function f at $(n+1)$ points and performing some elementary arithmetic. We describe this technique, along with a more accurate variant that requires additional function evaluations.

A popular formula for approximating the partial derivative $\partial f/\partial x_i$ at a given point x is the *forward-difference*, or *one-sided-difference*, approximation, defined as

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x+\epsilon e_i) - f(x)}{\epsilon}. \tag{8.1}$$

The gradient can be built up by simply applying this formula for $i = 1, 2, \ldots, n$. This process requires evaluation of f at the point x as well as the n perturbed points $x + \epsilon e_i$, $i = 1, 2, \ldots, n$: a total of $(n+1)$ points.

The basis for the formula (8.1) is Taylor's theorem, Theorem 2.1 in Chapter 2. When f is twice continuously differentiable, we have

$$f(x+p) = f(x) + \nabla f(x)^T p + \tfrac{1}{2} p^T \nabla^2 f(x+tp) p, \qquad \text{some } t \in (0,1) \tag{8.2}$$

(see (2.6)). If we choose L to be a bound on the size of $\|\nabla^2 f(\cdot)\|$ in the region of interest, it follows directly from this formula that the last term in this expression is bounded by $(L/2)\|p\|^2$, so that

$$\left\| f(x+p) - f(x) - \nabla f(x)^T p \right\| \leq (L/2)\|p\|^2. \tag{8.3}$$

We now choose the vector p to be ϵe_i, so that it represents a small change in the value of a single component of x (the ith component). For this p, we have that $\nabla f(x)^T p = \nabla f(x)^T e_i = \partial f/\partial x_i$, so by rearranging (8.3), we conclude that

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x+\epsilon e_i) - f(x)}{\epsilon} + \delta_\epsilon, \qquad \text{where } |\delta_\epsilon| \leq (L/2)\epsilon. \tag{8.4}$$

We derive the forward-difference formula (8.1) by simply ignoring the error term δ_ϵ in this expression, which becomes smaller and smaller as ϵ approaches zero.

An important issue in implementing the formula (8.1) is the choice of the parameter ϵ. The error expression (8.4) suggests that we should choose ϵ as small as possible. Unfortunately, this expression ignores the roundoff errors that are introduced when the function f is evaluated on a real computer, in floating-point arithmetic. From our discussion in the Appendix (see (A.30) and (A.31)), we know that the quantity **u** known as *unit roundoff*

is crucial: It is a bound on the relative error that is introduced whenever an arithmetic operation is performed on two floating-point numbers. ($\mathbf{u}$ is about 1.1×10^{-16} in double-precision IEEE floating-point arithmetic.) The effect of these errors on the final computed value of f depends on the way in which f is computed. It could come from an arithmetic formula, or from a differential equation solver, with or without refinement.

As a rough estimate, let us assume simply that the relative error in the computed f is bounded by $\mathbf{u}$, so that the computed values of $f(x)$ and $f(x + \epsilon e_i)$ are related to the exact values in the following way:

$$\begin{aligned} |\text{comp}(f(x)) - f(x)| &\leq \mathbf{u} L_f, \\ |\text{comp}(f(x + \epsilon e_i)) - f(x + \epsilon e_i)| &\leq \mathbf{u} L_f, \end{aligned}$$

where comp$(\cdot)$ denotes the computed value, and L_f is a bound on the value of $|f(\cdot)|$ in the region of interest. If we use these computed values of f in place of the exact values in (8.4) and (8.1), we obtain an error that is bounded by

$$(L/2)\epsilon + 2\mathbf{u} L_f/\epsilon. \tag{8.5}$$

Naturally, we would like to choose ϵ to make this error as small as possible; it is easy to see that the minimizing value is

$$\epsilon^2 = \frac{4 L_f \mathbf{u}}{L}.$$

If we assume that the problem is well scaled, then the ratio L_f/L (the ratio of function values to second derivative values) does not exceed a modest size. We can conclude that the following choice of ϵ is fairly close to optimal:

$$\epsilon = \sqrt{\mathbf{u}}. \tag{8.6}$$

(In fact, this value is used in many of the optimization software packages that use finite differencing as an option for estimating derivatives.) For this value of ϵ, we have from (8.5) that the total error in the forward-difference approximation is fairly close to $\sqrt{\mathbf{u}}$.

A more accurate approximation to the derivative can be obtained by using the *central-difference* formula, defined as

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon}. \tag{8.7}$$

As we show below, this approximation is more accurate than the forward-difference approximation (8.1). It is also about twice as expensive, since we need to evaluate f at the points x and $x \pm \epsilon e_i$, $i = 1, 2, \ldots, n$: a total of $2n + 1$ points.

The basis for the central difference approximation is again Taylor's theorem. When the second derivatives of f exist and are Lipschitz continuous, we have from (8.2) that

$$\begin{aligned} f(x+p) &= f(x) + \nabla f(x)^T p + \tfrac{1}{2} p^T \nabla^2 f(x+tp) p \quad \text{for some } t \in (0,1) \\ &= f(x) + \nabla f(x)^T p + \tfrac{1}{2} p^T \nabla^2 f(x) p + O\left(\|p\|^3\right). \end{aligned} \tag{8.8}$$

By setting $p = \epsilon e_i$ and $p = -\epsilon e_i$, respectively, we obtain

$$\begin{aligned} f(x+\epsilon e_i) &= f(x) + \epsilon \frac{\partial f}{\partial x_i} + \frac{1}{2}\epsilon^2 \frac{\partial^2 f}{\partial x_i^2} + O\left(\epsilon^3\right), \\ f(x-\epsilon e_i) &= f(x) - \epsilon \frac{\partial f}{\partial x_i} + \frac{1}{2}\epsilon^2 \frac{\partial^2 f}{\partial x_i^2} + O\left(\epsilon^3\right). \end{aligned}$$

(Note that the final error terms in these two expressions are generally not the same, but they are both bounded by some multiple of ϵ^3.) By subtracting the second equation from the first and dividing by 2ϵ, we obtain the expression

$$\frac{\partial f}{\partial x_i}(x) = \frac{f(x+\epsilon e_i) - f(x-\epsilon e_i)}{2\epsilon} + O\left(\epsilon^2\right).$$

We see from this expression that the error is $O\left(\epsilon^2\right)$, as compared to the $O(\epsilon)$ error in the forward-difference formula (8.1). However, when we take evaluation error in f into account, the accuracy that can be achieved in practice is less impressive; the same assumptions that were used to derive (8.6) lead to an optimal choice of ϵ of about $\mathbf{u}^{1/3}$ and an error of about $\mathbf{u}^{2/3}$. In some situations, the extra few digits of accuracy may improve the performance of the algorithm enough to make the extra expense worthwhile.

APPROXIMATING A SPARSE JACOBIAN

Consider now the case of a vector function $r : \mathbb{R}^n \to \mathbb{R}^m$, such as the residual vector that we consider in Chapter 10 or the system of nonlinear equations from Chapter 11. The matrix $J(x)$ of first derivatives for this function is defined as follows:

$$J(x) = \left[\frac{\partial r_j}{\partial x_i}\right]_{\substack{j=1,2,\ldots,m \\ i=1,2,\ldots,n}} = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}, \tag{8.9}$$

where r_j, $j = 1, 2, \ldots, m$ are the components of r. The techniques described in the previous

section can be used to evaluate the full Jacobian $J(x)$ one column at a time. When r is twice continuously differentiable, we can use Taylor's theorem to deduce that

$$\|r(x+p) - r(x) - J(x)p\| \leq (L/2)\|p\|^2, \tag{8.10}$$

where L is a Lipschitz constant for J in the region of interest. If we require an approximation to the Jacobian–vector product $J(x)p$ for a given vector p (as is the case with inexact Newton methods for nonlinear systems of equations; see Section 11.1), this expression immediately suggests choosing a small nonzero ϵ and setting

$$J(x)p \approx \frac{r(x+\epsilon p) - r(x)}{\epsilon}, \tag{8.11}$$

an approximation that is accurate to $O(\epsilon)$. A two-sided approximation can be derived from the formula (8.7).

If an approximation to the full Jacobian $J(x)$ is required, we can compute it a column at a time, analogously to (8.1), by setting set $p = \epsilon e_i$ in (8.10) to derive the following estimate of the ith column:

$$\frac{\partial r}{\partial x_i}(x) \approx \frac{r(x+\epsilon e_i) - r(x)}{\epsilon}. \tag{8.12}$$

A full Jacobian estimate can be obtained at a cost of $n+1$ evaluations of the function r. When the Jacobian is sparse, however, we can often obtain the estimate at a much lower cost, sometimes just three or four evaluations of r. The key is to estimate a number of different columns of the Jacobian simultaneously, by judicious choices of the perturbation vector p in (8.10).

We illustrate the technique with a simple example. Consider the function $r : \mathbb{R}^n \to \mathbb{R}^n$ defined by

$$r(x) = \begin{bmatrix} 2(x_2^3 - x_1^2) \\ 3(x_2^3 - x_1^2) + 2(x_3^3 - x_2^2) \\ 3(x_3^3 - x_2^2) + 2(x_4^3 - x_3^2) \\ \vdots \\ 3(x_n^3 - x_{n-1}^2) \end{bmatrix}. \tag{8.13}$$

Each component of r depends on just two or three components of x, so that each row of the Jacobian contains only two or three nonzero elements. For the case of $n = 6$, the Jacobian

has the following structure:

$$\begin{bmatrix} \times & \times & & & & & \\ \times & \times & \times & & & & \\ & \times & \times & \times & & & \\ & & \times & \times & \times & & \\ & & & \times & \times & \times & \\ & & & & \times & \times \end{bmatrix}, \tag{8.14}$$

where each cross represents a nonzero element, with zeros represented by a blank space.

Staying for the moment with the case $n = 6$, suppose that we wish to compute a finite-difference approximation to the Jacobian. (Of course, it is easy to calculate this particular Jacobian by hand, but there are complicated functions with similar structure for which hand calculation is more difficult.) A perturbation $p = \epsilon e_1$ to the first component of x will affect only the first and second components of r. The remaining components will be unchanged, so that the right-hand-side of formula (8.12) will correctly evaluate to zero in the components 3, 4, 5, 6. It is wasteful, however, to reevaluate these components of r when we know in advance that their values are not affected by the perturbation. Instead, we look for a way to modify the perturbation vector so that it does not have any further effect on components 1 and 2, but *does* produce a change in some of the components 3, 4, 5, 6, which we can then use as the basis of a finite-difference estimate for some *other* column of the Jacobian. It is not hard to see that the additional perturbation ϵe_4 has the desired property: It alters the 3rd, 4th, and 5th elements of r, but leaves the 1st and 2nd elements unchanged. The changes in r as a result of the perturbations ϵe_1 and ϵe_4 do not interfere with each other.

To express this discussion in mathematical terms, we set

$$p = \epsilon(e_1 + e_4),$$

and note that

$$r(x + p)_{1,2} = r(x + \epsilon(e_1 + e_4))_{1,2} = r(x + \epsilon e_1)_{1,2} \tag{8.15}$$

(where the notation $[\cdot]_{1,2}$ denotes the subvector consisting of the first and second elements), while

$$r(x + p)_{3,4,5} = r(x + \epsilon(e_1 + e_4))_{3,4,5} = r(x + \epsilon e_4)_{3,4,5}. \tag{8.16}$$

By substituting (8.15) into (8.10), we obtain

$$r(x + p)_{1,2} = r(x)_{1,2} + \epsilon[J(x)e_1]_{1,2} + O(\epsilon^2).$$

By rearranging this expression, we obtain the following difference formula for estimating the (1, 1) and (2, 1) elements of the Jacobian matrix:

$$\begin{bmatrix} \dfrac{\partial r_1}{\partial x_1}(x) \\ \dfrac{\partial r_2}{\partial x_1}(x) \end{bmatrix} = [J(x)e_1]_{1,2} \approx \frac{r(x+p)_{1,2} - r(x)_{1,2}}{\epsilon}. \tag{8.17}$$

A similar argument shows that the nonzero elements of the fourth column of the Jacobian can be estimated by substituting (8.16) into (8.10); we obtain

$$\begin{bmatrix} \dfrac{\partial r_4}{\partial x_3}(x) \\ \dfrac{\partial r_4}{\partial x_4}(x) \\ \dfrac{\partial r_4}{\partial x_5}(x) \end{bmatrix} = [J(x)e_4]_{3,4,5} \approx \frac{r(x+p)_{3,4,5} - r(x)_{3,4,5}}{\epsilon}. \tag{8.18}$$

To summarize: We have been able to estimate *two* columns of the Jacobian $J(x)$ by evaluating the function r at the single extra point $x + \epsilon(e_1 + e_4)$.

We can approximate the remainder of $J(x)$ in an economical manner as well. Columns 2 and 5 can be approximated by choosing $p = \epsilon(e_2 + e_5)$, while we can use $p = \epsilon(e_3 + e_6)$ to approximate columns 3 and 6. In total, we need 3 evaluations of the function r (after the initial evaluation at x) to estimate the entire Jacobian matrix.

In fact, for *any* choice of n in (8.13) (no matter how large), three extra evaluations of r are sufficient to approximate the entire Jacobian. The corresponding choices of perturbation vectors p are

$$\begin{aligned} p &= \epsilon(e_1 + e_4 + e_7 + e_{10} + \cdots), \\ p &= \epsilon(e_2 + e_5 + e_8 + e_{11} + \cdots), \\ p &= \epsilon(e_3 + e_6 + e_9 + e_{12} + \cdots). \end{aligned}$$

In the first of these vectors, the nonzero components are chosen so that no two of the columns $1, 4, 7, \ldots$ have a nonzero element in the same row. The same property holds for the other two vectors and, in fact, points the way to the criterion that we can apply to general problems to decide on a valid set of perturbation vectors.

Algorithms for choosing the perturbation vectors can be expressed conveniently in the language of graphs and graph coloring. For any function $r : \mathbb{R}^n \to \mathbb{R}^m$, we can construct a *column incidence graph* G with n nodes by drawing an arc between nodes i and k if there is some component of r that depends on both x_i and x_k. In other words, the ith and kth columns of the Jacobian $J(x)$ each have a nonzero element in some row j, for some $j = 1, 2, \ldots, m$ and some value of x. (The intersection graph for the function defined in

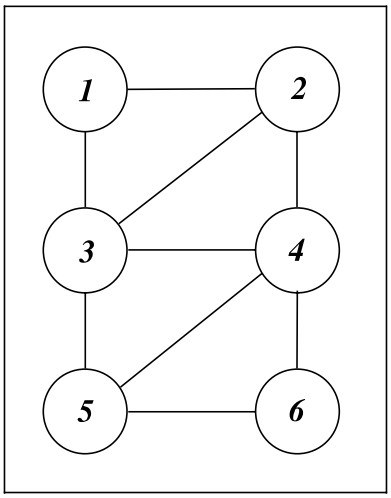


Figure 8.1
Column incidence graph for $r(x)$ defined in (8.13).

(8.13), with $n = 6$, is shown in Figure 8.1.) We now assign each node a "color" according to the following rule: Two nodes can have the same color if there is no arc that connects them. Finally, we choose one perturbation vector corresponding to each color: If nodes $i_1, i_2, \ldots, i_\ell$ have the same color, the corresponding p is $\epsilon(e_{i_1} + e_{i_2} + \cdots + e_{i_\ell})$.

Usually, there are many ways to assign colors to the n nodes in the graph in a way that satisfies the required condition. The simplest way is just to assign each node a different color, but since that scheme produces n perturbation vectors, it is usually not the most efficient approach. It is generally very difficult to find the coloring scheme that uses the fewest possible colors, but there are simple algorithms that do a good job of finding a near-optimal coloring at low cost. Curtis, Powell, and Reid [83] and Coleman and Moré [68] provide descriptions of some methods and performance comparisons. Newsam and Ramsdell [227] show that by considering a more general class of perturbation vectors p, it is possible to evaluate the full Jacobian using no more than n_z evaluations of r (in addition to the evaluation at the point x), where n_z is the maximum number of nonzeros in each row of $J(x)$.

For some functions r with well-studied structures (those that arise from discretizations of differential operators, or those that give rise to banded Jacobians, as in the example above), optimal coloring schemes are known. For the tridiagonal Jacobian of (8.14) and its associated graph in Figure 8.1, the scheme with three colors is optimal.

APPROXIMATING THE HESSIAN

In some situations, the user may be able to provide a routine to calculate the gradient $\nabla f(x)$ but not the Hessian $\nabla^2 f(x)$. We can obtain the Hessian by applying the techniques described above for the vector function r to the gradient ∇f. By using the graph coloring techniques discussed above, sparse Hessians often can be approximated in this manner by using considerably fewer than n perturbation vectors. This approach ignores symmetry of the Hessian, and will usually produce a nonsymmetric approximation. We can recover

symmetry by adding the approximation to its transpose and dividing the result by 2. Alternative differencing approaches that take symmetry of $\nabla^2 f(x)$ explicitly into account are discussed below.

Some important algorithms—most notably the Newton–CG methods described in Chapter 7—do not require knowledge of the full Hessian. Instead, each iteration requires us to supply the Hessian–vector product $\nabla^2 f(x)p$, for a given vector p. We can obtain an approximation to this matrix-vector product by appealing once again to Taylor's theorem. When second derivatives of f exist and are Lipschitz continuous near x, we have

$$\nabla f(x + \epsilon p) = \nabla f(x) + \epsilon \nabla^2 f(x)p + O(\epsilon^2), \tag{8.19}$$

so that

$$\nabla^2 f(x)p \approx \frac{\nabla f(x + \epsilon p) - \nabla f(x)}{\epsilon} \tag{8.20}$$

(see also (7.10)). The approximation error is $O(\epsilon)$, and the cost of obtaining the approximation is a single gradient evaluation at the point $x + \epsilon p$. The formula (8.20) corresponds to the forward-difference approximation (8.1). A central-difference formula like (8.7) can be derived by evaluating $\nabla f(x - \epsilon p)$ as well.

For the case in which even gradients are not available, we can use Taylor's theorem once again to derive formulae for approximating the Hessian that use only function values. The main tool is the formula (8.8): By substituting the vectors $p = \epsilon e_i$, $p = \epsilon e_j$, and $p = \epsilon(e_i + e_j)$ into this formula and combining the results appropriately, we obtain

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{f(x + \epsilon e_i + \epsilon e_j) - f(x + \epsilon e_i) - f(x + \epsilon e_j) + f(x)}{\epsilon^2} + O(\epsilon). \tag{8.21}$$

If we wished to approximate every element of the Hessian with this formula, then we would need to evaluate f at $x + \epsilon(e_i + e_j)$ for all possible i and j (a total of $n(n+1)/2$ points) as well as at the n points $x + \epsilon e_i$, $i = 1, 2, \ldots, n$. If the Hessian is sparse, we can, of course, reduce this operation count by skipping the evaluation whenever we know the element $\partial^2 f / \partial x_i \partial x_j$ to be zero.

APPROXIMATING A SPARSE HESSIAN

We noted above that a Hessian approximation can be obtained by applying finite-difference Jacobian estimation techniques to the gradient ∇f, treated as a vector function. We now show how symmetry of the Hessian $\nabla^2 f$ can be used to reduce the number of perturbation vectors p needed to obtain a complete approximation, when the Hessian is sparse. The key observation is that, because of symmetry, any estimate of the element $[\nabla^2 f(x)]_{i,j} = \partial^2 f(x)/\partial x_i \partial x_j$ is also an estimate of its symmetric counterpart $[\nabla^2 f(x)]_{j,i}$.

We illustrate the point with the simple function $f : \mathbb{R}^n \to \mathbb{R}$ defined by

$$f(x) = x_1 \sum_{i=1}^{n} i^2 x_i^2. \tag{8.22}$$

It is easy to show that the Hessian $\nabla^2 f$ has the "arrowhead" structure depicted below, for the case of $n = 6$:

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & & & \\ \times & & \times & & & \\ \times & & & \times & & \\ \times & & & & \times & \\ \times & & & & & \times \end{bmatrix}. \tag{8.23}$$

If we were to construct the intersection graph for the function ∇f (analogous to Figure 8.1), we would find that every node is connected to every other node, for the simple reason that row 1 has a nonzero in every column. According to the rule for coloring the graph, then, we would have to assign a different color to every node, which implies that we would need to evaluate ∇f at the $n + 1$ points x and $x + \epsilon e_i$ for $i = 1, 2, \ldots, n$.

We can construct a much more efficient scheme by taking the symmetry into account. Suppose we first use the perturbation vector $p = \epsilon e_1$ to estimate the first column of $\nabla^2 f(x)$. Because of symmetry, the same estimates apply to the first *row* of $\nabla^2 f$. From (8.23), we see that all that remains is to find the diagonal elements $\nabla^2 f(x)_{22}, \nabla^2 f(x)_{33}, \ldots, \nabla^2 f(x)_{66}$. The intersection graph for these remaining elements is completely disconnected, so we can assign them all the same color and choose the corresponding perturbation vector to be

$$p = \epsilon(e_2 + e_3 + \cdots + e_6) = \epsilon(0, 1, 1, 1, 1, 1)^T. \tag{8.24}$$

Note that the second component of ∇f is not affected by the perturbations in components 3, 4, 5, 6 of the unknown vector, while the third component of ∇f is not affected by perturbations in components 2, 4, 5, 6 of x, and so on. As in (8.15) and (8.16), we have for each component i that

$$\nabla f(x + p)_i = \nabla f(x + \epsilon(e_2 + e_3 + \cdots + e_6))_i = \nabla f(x + \epsilon e_i)_i.$$

By applying the forward-difference formula (8.1) to each of these individual components, we then obtain

$$\frac{\partial^2 f}{\partial x_i^2}(x) \approx \frac{\nabla f(x + \epsilon e_i)_i - \nabla f(x)_i}{\epsilon} = \frac{\nabla f(x + \epsilon p)_i - \nabla f(x)_i}{\epsilon}, \qquad i = 2, 3, \ldots, 6.$$

By exploiting symmetry, we have been able to estimate the entire Hessian by evaluating ∇f only at x and two other points.

Again, graph-coloring techniques can be used to choose the perturbation vectors p economically. We use the *adjacency graph* in place of the intersection graph described earlier. The adjacency graph has n nodes, with arcs connecting nodes i and k whenever $i \neq k$ and $\partial^2 f(x)/(\partial x_i \partial x_k) \neq 0$ for some x. The requirements on the coloring scheme are a little more complicated than before, however. We require not only that connected nodes have different colors, but also that any path of length 3 through the graph contain at least three colors. In other words, if there exist nodes i_1, i_2, i_3, i_4 in the graph that are connected by arcs (i_1, i_2), (i_2, i_3), and (i_3, i_4), then at least three different colors must be used in coloring these four nodes. See Coleman and Moré [69] for an explanation of this rule and for algorithms to compute valid colorings. The perturbation vectors are constructed as before: Whenever the nodes $i_1, i_2, \ldots, i_\ell$ have the same color, we set the corresponding perturbation vector to be $p = \epsilon(e_{i_1} + e_{i_2} + \cdots + e_{i_\ell})$.

8.2 AUTOMATIC DIFFERENTIATION

Automatic differentiation is the generic name for techniques that use the computational representation of a function to produce analytic values for the derivatives. Some techniques produce code for the derivatives at a general point x by manipulating the function code directly. Other techniques keep a record of the computations made during the evaluation of the function at a specific point x and then review this information to produce a set of derivatives at x.

Automatic differentiation techniques are founded on the observation that any function, no matter how complicated, is evaluated by performing a sequence of simple elementary operations involving just one or two arguments at a time. Two-argument operations include addition, multiplication, division, and the power operation a^b. Examples of single-argument operations include the trigonometric, exponential, and logarithmic functions. Another common ingredient of the various automatic differentiation tools is their use of the *chain rule*. This is the well-known rule from elementary calculus that says that if h is a function of the vector $y \in \mathbb{R}^m$, which is in turn a function of the vector $x \in \mathbb{R}^n$, we can write the derivative of h with respect to x as follows:

$$\nabla_x h(y(x)) = \sum_{i=1}^{m} \frac{\partial h}{\partial y_i} \nabla y_i(x). \tag{8.25}$$

See Appendix A for further details.

There are two basic modes of automatic differentiation: the *forward* and *reverse* modes. The difference between them can be illustrated by a simple example. We work through such

CHAPTER 12

Theory of Constrained Optimization

The second part of this book is about minimizing functions subject to constraints on the variables. A general formulation for these problems is

$$\min_{x \in \mathbb{R}^n} f(x) \qquad \text{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E}, \\ c_i(x) \geq 0, & i \in \mathcal{I}, \end{cases} \tag{12.1}$$

where f and the functions c_i are all smooth, real-valued functions on a subset of $\mathbb{R}^n$, and $\mathcal{I}$ and $\mathcal{E}$ are two finite sets of indices. As before, we call f the *objective* function, while c_i,

$i \in \mathcal{E}$ are the *equality constraints* and c_i, $i \in \mathcal{I}$ are the *inequality constraints*. We define the *feasible set* Ω to be the set of points x that satisfy the constraints; that is,

$$\Omega = \{x \mid c_i(x) = 0, \quad i \in \mathcal{E}; \quad c_i(x) \geq 0, \quad i \in \mathcal{I}\}, \tag{12.2}$$

so that we can rewrite (12.1) more compactly as

$$\min_{x \in \Omega} \; f(x). \tag{12.3}$$

In this chapter we derive mathematical characterizations of the solutions of (12.3). As in the unconstrained case, we discuss optimality conditions of two types. *Necessary* conditions are conditions that must be satisfied by any solution point (under certain assumptions). *Sufficient* conditions are those that, if satisfied at a certain point x^*, guarantee that x^* is in fact a solution.

For the unconstrained optimization problem of Chapter 2, the optimality conditions were as follows:

> Necessary conditions: Local unconstrained minimizers have $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ positive semidefinite.
>
> Sufficient conditions: Any point x^* at which $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite is a strong local minimizer of f.

In this chapter, we derive analogous conditions to characterize the solutions of constrained optimization problems.

LOCAL AND GLOBAL SOLUTIONS

We have seen already that global solutions are difficult to find even when there are no constraints. The situation may be improved when we add constraints, since the feasible set might exclude many of the local minima and it may be comparatively easy to pick the global minimum from those that remain. However, constraints can also make things more difficult. As an example, consider the problem

$$\min (x_2 + 100)^2 + 0.01x_1^2, \qquad \text{subject to } x_2 - \cos x_1 \geq 0, \tag{12.4}$$

illustrated in Figure 12.1. Without the constraint, the problem has the unique solution $(0, -100)^T$. With the constraint, there are local solutions near the points

$$x^{(k)} = (k\pi, -1)^T, \qquad \text{for } k = \pm 1, \pm 3, \pm 5, \ldots.$$

Definitions of the different types of local solutions are simple extensions of the corresponding definitions for the unconstrained case, except that now we restrict consideration to the *feasible* points in the neighborhood of x^*. We have the following definition.

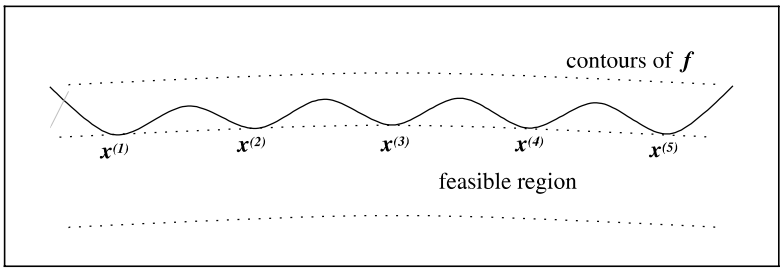


Figure 12.1 Constrained problem with many isolated local solutions.

A vector x^* is a *local solution* of the problem (12.3) if $x^* \in \Omega$ and there is a neighborhood $\mathcal{N}$ of x^* such that $f(x) \geq f(x^*)$ for $x \in \mathcal{N} \cap \Omega$.

Similarly, we can make the following definitions:

A vector x^* is a *strict local solution* (also called a *strong local solution*) if $x^* \in \Omega$ and there is a neighborhood $\mathcal{N}$ of x^* such that $f(x) > f(x^*)$ for all $x \in \mathcal{N} \cap \Omega$ with $x \neq x^*$.

A point x^* is an *isolated local solution* if $x^* \in \Omega$ and there is a neighborhood $\mathcal{N}$ of x^* such that x^* is the only local solution in $\mathcal{N} \cap \Omega$.

Note that isolated local solutions are strict, but that the reverse is not true (see Exercise 12.2).

SMOOTHNESS

Smoothness of objective functions and constraints is an important issue in characterizing solutions, just as in the unconstrained case. It ensures that the objective function and the constraints all behave in a reasonably predictable way and therefore allows algorithms to make good choices for search directions.

We saw in Chapter 2 that graphs of nonsmooth functions contain "kinks" or "jumps" where the smoothness breaks down. If we plot the feasible region for any given constrained optimization problem, we usually observe many kinks and sharp edges. Does this mean that the constraint functions that describe these regions are nonsmooth? The answer is often no, because the nonsmooth boundaries can often be described by a collection of smooth constraint functions. Figure 12.2 shows a diamond-shaped feasible region in $\mathbb{R}^2$ that could be described by the single nonsmooth constraint

$$\|x\|_1 = |x_1| + |x_2| \leq 1. \tag{12.5}$$

It can also be described by the following set of smooth (in fact, linear) constraints:

$$x_1 + x_2 \leq 1, \quad x_1 - x_2 \leq 1, \quad -x_1 + x_2 \leq 1, \quad -x_1 - x_2 \leq 1. \tag{12.6}$$

Each of the four constraints represents one edge of the feasible polytope. In general, the constraint functions are chosen so that each one represents a smooth piece of the boundary of Ω.

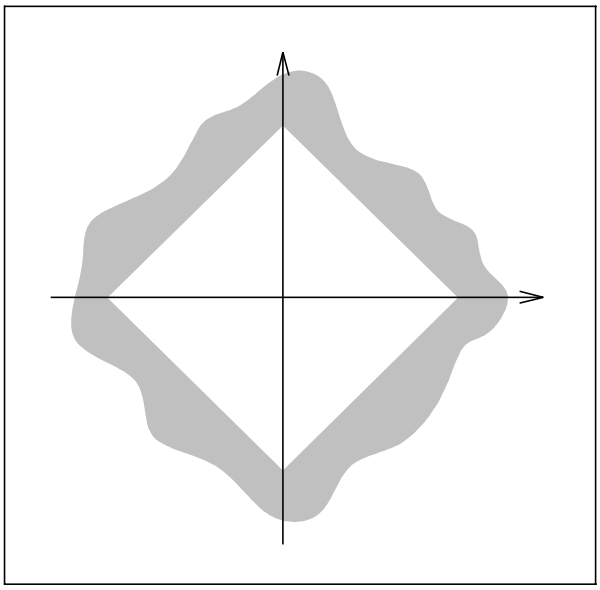


Figure 12.2
A feasible region with a nonsmooth boundary can be described by smooth constraints.

Nonsmooth, unconstrained optimization problems can sometimes be reformulated as smooth constrained problems. An example is the unconstrained minimization of a function

$$f(x) = \max(x^2, x), \tag{12.7}$$

which has kinks at $x = 0$ and $x = 1$, and the solution at $x^* = 0$. We obtain a smooth, constrained formulation of this problem by adding an artificial variable t and writing

$$\min t \qquad \text{s.t.} \qquad t \geq x, \quad t \geq x^2. \tag{12.8}$$

Reformulation techniques such as (12.6) and (12.8) are used often in cases where f is a maximum of a collection of functions or when f is a 1-norm or ∞-norm of a vector function.

In the examples above we expressed inequality constraints in a slightly different way from the form $c_i(x) \geq 0$ that appears in the definition (12.1). However, any collection of inequality constraints with $\geq$ and $\leq$ and nonzero right-hand-sides can be expressed in the form $c_i(x) \geq 0$ by simple rearrangement of the inequality.

12.1 EXAMPLES

To introduce the basic principles behind the characterization of solutions of constrained optimization problems, we work through three simple examples. The discussion here is informal; the ideas introduced will be made rigorous in the sections that follow.

We start by noting one important item of terminology that recurs throughout the rest of the book.

Definition 12.1.

The active set $\mathcal{A}(x)$ *at any feasible* x *consists of the equality constraint indices from* $\mathcal{E}$ *together with the indices of the inequality constraints* i *for which* $c_i(x) = 0$*; that is,*

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\}.$$

At a feasible point x, the inequality constraint $i \in \mathcal{I}$ is said to be *active* if $c_i(x) = 0$ and *inactive* if the strict inequality $c_i(x) > 0$ is satisfied.

A SINGLE EQUALITY CONSTRAINT

❑ EXAMPLE 12.1

Our first example is a two-variable problem with a single equality constraint:

$$\min\ x_1 + x_2 \qquad \text{s.t.} \qquad x_1^2 + x_2^2 - 2 = 0 \tag{12.9}$$

(see Figure 12.3). In the language of (12.1), we have $f(x) = x_1 + x_2$, $\mathcal{I} = \emptyset$, $\mathcal{E} = \{1\}$, and $c_1(x) = x_1^2 + x_2^2 - 2$. We can see by inspection that the feasible set for this problem is the circle of radius $\sqrt{2}$ centered at the origin—just the boundary of this circle, not its interior. The solution x^* is obviously $(-1, -1)^T$. From any other point on the circle, it is easy to find a way to move that *stays feasible* (that is, remains on the circle) while *decreasing* f. For instance, from the point $x = (\sqrt{2}, 0)^T$ any move in the clockwise direction around the circle has the desired effect.

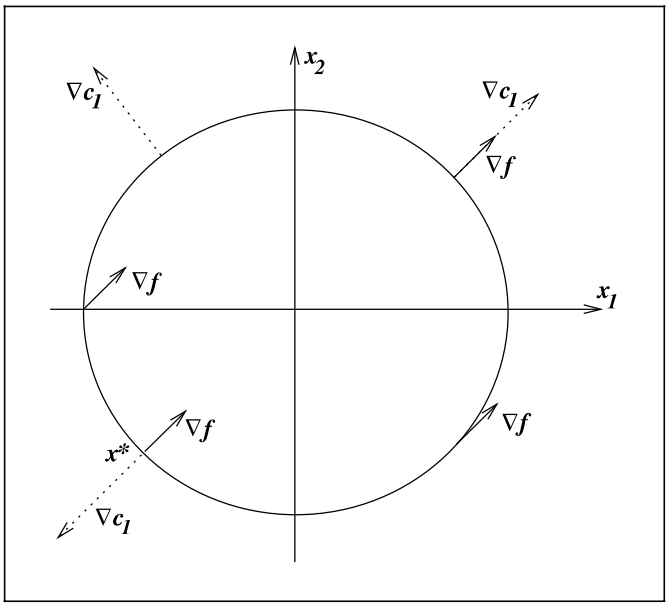


Figure 12.3
Problem (12.9), showing constraint and function gradients at various feasible points.

We also see from Figure 12.3 that at the solution x^*, the *constraint normal* $\nabla c_1(x^*)$ is parallel to $\nabla f(x^*)$. That is, there is a scalar λ_1^* (in this case $\lambda_1^* = -1/2$) such that

$$\nabla f(x^*) = \lambda_1^* \nabla c_1(x^*). \tag{12.10}$$

❒

We can derive (12.10) by examining first-order Taylor series approximations to the objective and constraint functions. To retain feasibility with respect to the function $c_1(x) = 0$, we require any small (but nonzero) step s to satisfy that $c_1(x + s) = 0$; that is,

$$0 = c_1(x + s) \approx c_1(x) + \nabla c_1(x)^T s = \nabla c_1(x)^T s. \tag{12.11}$$

Hence, the step s retains feasibility with respect to c_1, to first order, when it satisfies

$$\nabla c_1(x)^T s = 0. \tag{12.12}$$

Similarly, if we want s to produce a decrease in f, we would have so that

$$0 > f(x + s) - f(x) \approx \nabla f(x)^T s,$$

or, to first order,

$$\nabla f(x)^T s < 0. \tag{12.13}$$

Existence of a small step s that satisfies both (12.12) and (12.13) strongly suggests existence of a direction d (where the size of d is *not* small; we could have $d \approx s/\|s\|$ to ensure that the norm of d is close to 1) with the same properties, namely

$$\nabla c_1(x)^T d = 0 \quad \text{and} \quad \nabla f(x)^T d < 0. \tag{12.14}$$

If, on the other hand, there is *no* direction d with the properties (12.14), then is it likely that we cannot find a small step s with the properties (12.12) and (12.13). In this case, x^* would appear to be a local minimizer.

By drawing a picture, the reader can check that the only way that a d satisfying (12.14) does *not* exist is if $\nabla f(x)$ and $\nabla c_1(x)$ are parallel, that is, if the condition $\nabla f(x) = \lambda_1 \nabla c_1(x)$ holds at x, for some scalar λ_1. If in fact $\nabla f(x)$ and $\nabla c_1(x)$ are *not* parallel, we can set

$$\bar{d} = -\left(I - \frac{\nabla c_1(x) \nabla c_1(x)^T}{\|\nabla c_1(x)\|^2}\right) \nabla f(x); \qquad d = \frac{\bar{d}}{\|\bar{d}\|}. \tag{12.15}$$

It is easy to verify that this d satisfies (12.14).

By introducing the *Lagrangian function*

$$\mathcal{L}(x, \lambda_1) = f(x) - \lambda_1 c_1(x), \tag{12.16}$$

and noting that $\nabla_x \mathcal{L}(x, \lambda_1) = \nabla f(x) - \lambda_1 \nabla c_1(x)$, we can state the condition (12.10) equivalently as follows: At the solution x^*, there is a scalar λ_1^* such that

$$\nabla_x \mathcal{L}(x^*, \lambda_1^*) = 0. \tag{12.17}$$

This observation suggests that we can search for solutions of the equality-constrained problem (12.9) by seeking stationary points of the Lagrangian function. The scalar quantity λ_1 in (12.16) is called a *Lagrange multiplier* for the constraint $c_1(x) = 0$.

Though the condition (12.10) (equivalently, (12.17)) appears to be *necessary* for an optimal solution of the problem (12.9), it is clearly not *sufficient*. For instance, in Example 12.1, condition (12.10) is satisfied at the point $x = (1, 1)^T$ (with $\lambda_1 = \frac{1}{2}$), but this point is obviously not a solution—in fact, it *maximizes* the function f on the circle. Moreover, in the case of equality-constrained problems, we cannot turn the condition (12.10) into a sufficient condition simply by placing some restriction on the sign of λ_1. To see this, consider replacing the constraint $x_1^2 + x_2^2 - 2 = 0$ by its negative $2 - x_1^2 - x_2^2 = 0$ in Example 12.1. The solution of the problem is not affected, but the value of λ_1^* that satisfies the condition (12.10) changes from $\lambda_1^* = -\frac{1}{2}$ to $\lambda_1^* = \frac{1}{2}$.

A SINGLE INEQUALITY CONSTRAINT

❑ EXAMPLE 12.2

This is a slight modification of Example 12.1, in which the equality constraint is replaced by an inequality. Consider

$$\min\ x_1 + x_2 \qquad \text{s.t.} \qquad 2 - x_1^2 - x_2^2 \geq 0, \tag{12.18}$$

for which the feasible region consists of the circle of problem (12.9) and its interior (see Figure 12.4). Note that the constraint normal ∇c_1 points toward the interior of the feasible region at each point on the boundary of the circle. By inspection, we see that the solution is still $(-1, -1)^T$ and that the condition (12.10) holds for the value $\lambda_1^* = \frac{1}{2}$. However, this inequality-constrained problem differs from the equality-constrained problem (12.9) of Example 12.1 in that the sign of the Lagrange multiplier plays a significant role, as we now argue.

❐

As before, we conjecture that a given feasible point x is *not* optimal if we can find a small step s that both retains feasibility and decreases the objective function f to first order. The main difference between problems (12.9) and (12.18) comes in the handling of the feasibility condition. As in (12.13), the step s improves the objective function, to first order, if $\nabla f(x)^T s < 0$. Meanwhile, s retains feasibility if

$$0 \leq c_1(x+s) \approx c_1(x) + \nabla c_1(x)^T s,$$

so, to first order, feasibility is retained if

$$c_1(x) + \nabla c_1(x)^T s \geq 0. \tag{12.19}$$

In determining whether a step s exists that satisfies both (12.13) and (12.19), we consider the following two cases, which are illustrated in Figure 12.4.

Case I: Consider first the case in which x lies *strictly inside* the circle, so that the strict inequality $c_1(x) > 0$ holds. In this case, *any* step vector s satisfies the condition (12.19), provided only that its length is sufficiently small. In fact, whenever $\nabla f(x) \neq 0$, we can obtain a step s that satisfies both (12.13) and (12.19) by setting

$$s = -\alpha \nabla f(x),$$

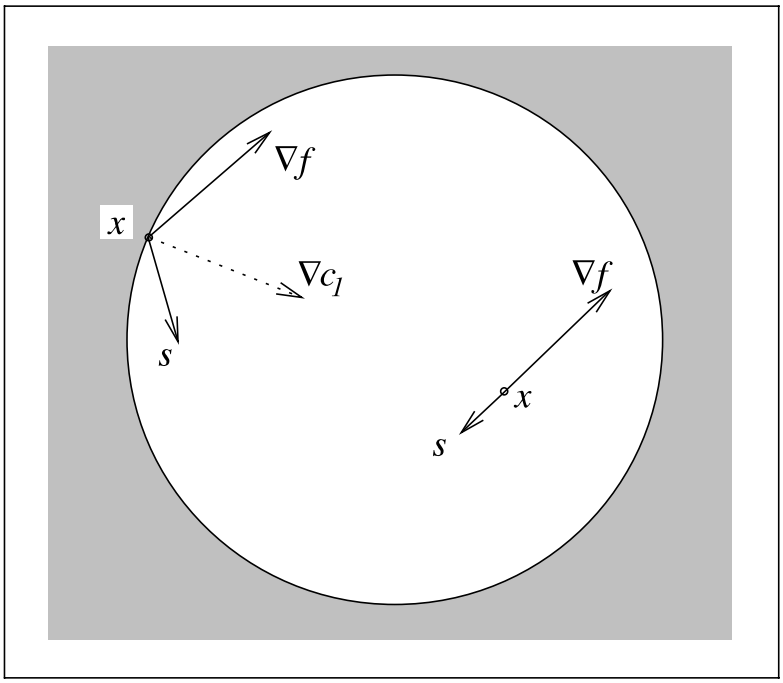


Figure 12.4 Improvement directions s from two feasible points x for the problem (12.18) at which the constraint is active and inactive, respectively.

for any positive scalar α sufficiently small. However, this definition does not give a step s with the required properties when

$$\nabla f(x) = 0, \tag{12.20}$$

Case II: Consider now the case in which x lies on the boundary of the circle, so that $c_1(x) = 0$. The conditions (12.13) and (12.19) therefore become

$$\nabla f(x)^T s < 0, \qquad \nabla c_1(x)^T s \geq 0.$$

The first of these conditions defines an open half-space, while the second defines a closed half-space, as illustrated in Figure 12.5. It is clear from this figure that the intersection of these two regions is empty only when $\nabla f(x)$ and $\nabla c_1(x)$ point in the same direction, that is, when

$$\nabla f(x) = \lambda_1 \nabla c_1(x), \qquad \text{for some } \lambda_1 \geq 0. \tag{12.21}$$

Note that the sign of the multiplier is significant here. If (12.10) were satisfied with a *negative* value of λ_1, then $\nabla f(x)$ and $\nabla c_1(x)$ would point in opposite directions, and we see from Figure 12.5 that the set of directions that satisfy both (12.13) and (12.19) would make up an entire open half-plane.

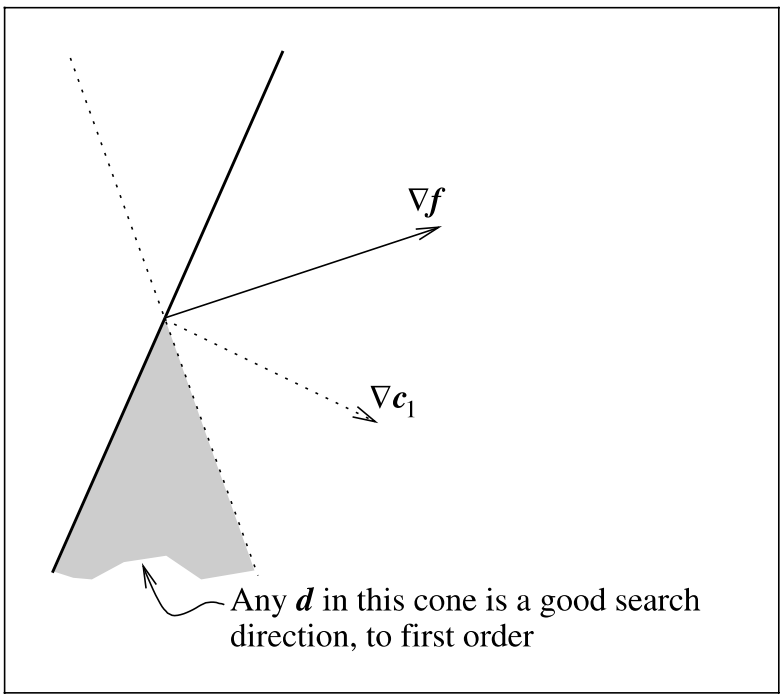


Figure 12.5 A direction d that satisfies both (12.13) and (12.19) lies in the intersection of a closed half-plane and an open half-plane.

The optimality conditions for both cases I and II can again be summarized neatly with reference to the Lagrangian function $\mathcal{L}$ defined in (12.16). When no first-order feasible descent direction exists at some point x^*, we have that

$$\nabla_x \mathcal{L}(x^*, \lambda_1^*) = 0, \quad \text{for some } \lambda_1^* \geq 0, \tag{12.22}$$

where we also require that

$$\lambda_1^* c_1(x^*) = 0. \tag{12.23}$$

Condition (12.23) is known as a *complementarity condition*; it implies that the Lagrange multiplier λ_1 can be strictly positive *only when the corresponding constraint* c_1 *is active.* Conditions of this type play a central role in constrained optimization, as we see in the sections that follow. In case I, we have that $c_1(x^*) > 0$, so (12.23) requires that $\lambda_1^* = 0$. Hence, (12.22) reduces to $\nabla f(x^*) = 0$, as required by (12.20). In case II, (12.23) allows λ_1^* to take on a nonnegative value, so (12.22) becomes equivalent to (12.21).

TWO INEQUALITY CONSTRAINTS

❑ EXAMPLE 12.3

Suppose we add an extra constraint to the problem (12.18) to obtain

$$\min\ x_1 + x_2 \qquad \text{s.t.} \qquad 2 - x_1^2 - x_2^2 \geq 0, \quad x_2 \geq 0, \tag{12.24}$$

for which the feasible region is the half-disk illustrated in Figure 12.6. It is easy to see that the solution lies at $(-\sqrt{2}, 0)^T$, a point at which both constraints are active. By repeating the arguments for the previous examples, we would expect a direction d of first-order feasible descent to satisfy

$$\nabla c_i(x)^T d \geq 0, \quad i \in \mathcal{I} = \{1, 2\}, \quad \nabla f(x)^T d < 0. \tag{12.25}$$

However, it is clear from Figure 12.6 that no such direction can exist when $x = (-\sqrt{2}, 0)^T$. The conditions $\nabla c_i(x)^T d \geq 0$, $i = 1, 2$, are both satisfied only if d lies in the quadrant defined by $\nabla c_1(x)$ and $\nabla c_2(x)$, but it is clear by inspection that all vectors d in this quadrant satisfy $\nabla f(x)^T d \geq 0$.

Let us see how the Lagrangian and its derivatives behave for the problem (12.24) and the solution point $(-\sqrt{2}, 0)^T$. First, we include an additional term $\lambda_i c_i(x)$ in the Lagrangian for each additional constraint, so the definition of $\mathcal{L}$ becomes

$$\mathcal{L}(x, \lambda) = f(x) - \lambda_1 c_1(x) - \lambda_2 c_2(x),$$