

# OPTIMIZATION FOR ENGINEERING DESIGN



# **Optimization for Engineering Design**

## Algorithms and Examples

---

SECOND EDITION

---

**KALYANMOY DEB**

*Department of Mechanical Engineering  
Indian Institute of Technology Kanpur*

**PHI Learning Private Limited**  
New Delhi-110001  
2012

**OPTIMIZATION FOR ENGINEERING DESIGN—Algorithms and Examples, Second Edition**  
Kalyanmoy Deb

© 2012 by PHI Learning Private Limited, New Delhi. All rights reserved. No part of this book may be reproduced in any form, by mimeograph or any other means, without permission in writing from the publisher.

**ISBN-978-81-203-4678-9**

The export rights of this book are vested solely with the publisher.

**Twelfth Printing (Second Edition)** ... ... **November, 2012**

Published by Asoke K. Ghosh, PHI Learning Private Limited, M-97, Connaught Circus, New Delhi-110001 and Printed by Rajkamal Electric Press, Plot No. 2, Phase IV, HSIDC, Kundli-131028, Sonepat, Haryana.

To  
*My Parents*



# Contents

---

<i>Preface</i> .....	<i>xi</i>
<i>Preface to the First Edition</i> .....	<i>xiii</i>
<i>Acknowledgements</i> .....	<i>xvii</i>

## 1. Introduction..... 1–42

1.1 Optimal Problem Formulation	2
1.1.1 Design Variables	3
1.1.2 Constraints	4
1.1.3 Objective Function	5
1.1.4 Variable Bounds	6
1.2 Engineering Optimization Problems	8
1.2.1 Design and Manufacturing	9
1.2.2 Modelling	17
1.2.3 Data Fitting and Regression	21
1.2.4 Control Systems	22
1.2.5 Inverse Problems	24
1.2.6 Scheduling and Routing	26
1.2.7 Data Mining	31
1.2.8 Intelligent System Design	32
1.3 Classification of Optimization Algorithms	35
1.4 Summary	40
<i>References</i>	40

## 2. Single-variable Optimization Algorithms ..... 43–84

2.1 Optimality Criteria	44
2.2 Bracketing Methods	46
2.2.1 Exhaustive Search Method	46
2.2.2 Bounding Phase Method	49
2.3 Region-Elimination Methods	51
2.3.1 Interval Halving Method	52

2.3.2	Fibonacci Search Method	<b>54</b>
2.3.3	Golden Section Search Method	<b>57</b>
2.4	Point-Estimation Method	<b>60</b>
2.4.1	Successive Quadratic Estimation Method	<b>60</b>
2.5	Gradient-based Methods	<b>63</b>
2.5.1	Newton-Raphson Method	<b>63</b>
2.5.2	Bisection Method	<b>65</b>
2.5.3	Secant Method	<b>67</b>
2.5.4	Cubic Search Method	<b>68</b>
2.6	Root-finding Using Optimization Techniques	<b>71</b>
2.7	Summary	<b>74</b>
<i>References</i> <b>75</b>		
<i>Problems</i> <b>75</b>		
<i>Computer Programs</i> <b>78</b>		
<b>3.</b>	<b>Multivariable Optimization Algorithms</b>	<b>85–142</b>
3.1	Optimality Criteria	<b>85</b>
3.2	Unidirectional Search	<b>87</b>
3.3	Direct Search Methods	<b>89</b>
3.3.1	Box's Evolutionary Optimization Method	<b>90</b>
3.3.2	Simplex Search Method	<b>95</b>
3.3.3	Hooke-Jeeves Pattern Search Method	<b>98</b>
3.3.4	Powell's Conjugate Direction Method	<b>103</b>
3.4	Gradient-based Methods	<b>108</b>
3.4.1	Cauchy's (Steepest Descent) Method	<b>112</b>
3.4.2	Newton's Method	<b>114</b>
3.4.3	Marquardt's Method	<b>118</b>
3.4.4	Conjugate Gradient Method	<b>120</b>
3.4.5	Variable-metric Method (DFP Method)	<b>124</b>
3.5	Summary	<b>128</b>
<i>References</i> <b>129</b>		
<i>Problems</i> <b>130</b>		
<i>Computer Program</i> <b>134</b>		
<b>4.</b>	<b>Constrained Optimization Algorithms</b>	<b>143–262</b>
4.1	Kuhn-Tucker Conditions	<b>144</b>
4.2	Lagrangian Duality Theory	<b>151</b>
4.3	Transformation Methods	<b>154</b>
4.3.1	Penalty Function Method	<b>154</b>
4.3.2	Method of Multipliers	<b>162</b>
4.4	Sensitivity Analysis	<b>167</b>
4.5	Direct Search for Constrained Minimization	<b>173</b>
4.5.1	Variable Elimination Method	<b>173</b>
4.5.2	Complex Search Method	<b>177</b>
4.5.3	Random Search Methods	<b>182</b>

4.6	Linearized Search Techniques	185
4.6.1	Frank-Wolfe Method	186
4.6.2	Cutting Plane Method	192
4.7	Feasible Direction Method	203
4.8	Quadratic Programming	212
4.8.1	Sequential Quadratic Programming	218
4.9	Generalized Reduced Gradient Method	224
4.10	Gradient Projection Method	232
4.11	Summary	239
	References	242
	Problems	243
	Computer Program	253
<b>5.</b>	<b>Specialized Algorithms</b>	<b>263–291</b>
5.1	Integer Programming	264
5.1.1	Penalty Function Method	265
5.1.2	Branch-and-Bound Method	270
5.2	Geometric Programming	278
5.3	Summary	288
	References	288
	Problems	289
<b>6.</b>	<b>Nontraditional Optimization Algorithms</b>	<b>292–368</b>
6.1	Genetic Algorithms	292
6.1.1	Working Principles	293
6.1.2	Differences between GAs and Traditional Methods	298
6.1.3	Similarities between GAs and Traditional Methods	301
6.1.4	GAs for Constrained Optimization	311
6.1.5	Other GA Operators	314
6.1.6	Real-coded GAs	315
6.1.7	Multi-objective GAs	319
6.1.8	Other Advanced GAs	323
6.2	Simulated Annealing	325
6.3	Global Optimization	330
6.3.1	Using the Steepest Descent Method	330
6.3.2	Using Genetic Algorithms	332
6.3.3	Using Simulated Annealing	334
6.4	Summary	335
	References	336
	Problems	340
	Computer Program	348
	<i>Appendix: Linear Programming Algorithms</i>	369–415
	<i>Index</i>	417–421



# Preface

---

The first edition of this book which was published in 1995 has been well tested at IIT Kanpur and at many other universities over the past 17 years. It is unusual to have the second edition of a book being published after so many years, but it is the nature of the book that prompted me to wait till there is enough feedback from students and teachers before I was sitting down to revise the first edition. The optimization algorithms laid out in this book do not change with time, although their explanations and presentations could have been made better. But the feedback I received from several of my students and a large number of instructors has been positive and I had not much motivation to revise the book in a major way. The simplified presentation of optimization algorithms remains as a hallmark feature of this book. Purposefully, a few topics of optimization were left out in the first edition, which I have now included in this edition. Specifically, a section on quadratic programming and its extension to sequential quadratic programming have been added. Genetic algorithms (GAs) for optimization have been significantly modified in the past 17 years, but if I have to provide an account of all the current methods of GAs, it will be a book of its own. But I could not resist to include some details on real-parameter GAs and multi-objective optimization. Readers interested in knowing more about GAs are encouraged to refer to most recent books and conference proceedings on the topic.

A major modification has been made to the Linear Programming (LP) chapter in the Appendix. Several methods including sensitivity analysis procedures have been added so that students can get a comprehensive idea of different LP methods. While making the modifications, the simplicity of the algorithms, as it was presented in the first edition, has been kept. Finally, more exercise problems are added not only to this chapter, but to all previous chapters of this revised book.

I sincerely hope the second edition becomes more useful in getting a proper understanding of different optimization algorithms discussed in this book. I would appreciate very much if any typing error or comments can be directly sent to my email address: deb@iitk.ac.in or kalyanmoy.deb@gmail.com.

**Kalyanmoy Deb**

# Preface to the First Edition

---

Many engineers and researchers in industries and academics face difficulty in understanding the role of optimization in engineering design. To many of them, optimization is an esoteric technique used only in mathematics and operations research related activities. With the advent of computers, optimization has become a part of computer-aided design activities. It is primarily being used in those design activities in which the goal is not only to achieve just a feasible design, but also a design objective. In most engineering design activities, the design objective could be simply to minimize the cost of production or to maximize the efficiency of production. An optimization algorithm is a procedure which is executed iteratively by comparing various solutions till the optimum or a satisfactory solution is found. In many industrial design activities, optimization is achieved indirectly by comparing a few chosen design solutions and accepting the best solution. This simplistic approach never guarantees an optimal solution. On the contrary, optimization algorithms begin with one or more design solutions supplied by the user and then iteratively check new design solutions in order to achieve the true optimum solution. In this book, I have put together and discussed a few popular optimization algorithms and demonstrated their working principles by hand-simulating on a simple example problem. Some working computer codes are also appended for limited use.

There are two distinct types of optimization algorithms which are in use today. First, there are algorithms which are deterministic, with specific rules for moving from one solution to the other. These algorithms have been in use for quite some time and have been successfully applied to many engineering design problems. Secondly, there are algorithms which are stochastic in nature, with probabilistic transition rules. These algorithms are comparatively new and are gaining popularity due to certain properties which the deterministic algorithms do not have. In this book, probably for the first time, an attempt has been made to present both these types

of algorithms in a single volume. Because of the growing complexity in engineering design problems, the designer can no longer afford to rely on a particular method. The designer must know the advantages and limitations of various methods and choose the one that is more efficient to the problem at hand.

An important aspect of the optimal design process is the formulation of the design problem in a mathematical format which is acceptable to an optimization algorithm. However, there is no unique way of formulating every engineering design problem. To illustrate the variations encountered in the formulation process, I have presented four different engineering design problems in Chapter 1. Optimization problems usually contain multiple design variables, but I have begun by first presenting a number of single-variable function optimization algorithms in Chapter 2. The working principles of these algorithms are simpler and, therefore, easier to understand. Besides, these algorithms are used in multivariable optimization algorithms as unidirectional search methods. Chapter 3 presents a number of algorithms for optimizing unconstrained objective functions having multiple variables. Chapter 4 is an important one in that it discusses a number of algorithms for solving constrained optimization problems—most engineering design optimization problems are constrained. Chapter 5 deals with two specialized algorithms for solving integer programming problems and geometric programming problems. Two nontraditional optimization algorithms, which are very different in principle than the above algorithms, are covered in Chapter 6. Genetic algorithms—search and optimization algorithms that mimic natural evolution and genetics—are potential optimization algorithms and have been applied to many engineering design problems in the recent past. Due to their population approach and parallel processing, these algorithms have been able to obtain global optimal solutions in complex optimization problems. Simulated annealing method mimics the cooling phenomenon of molten metals. Due to its inherent stochastic approach and availability of a convergence proof, this technique has also been used in many engineering design problems. Chapter 6 also discusses the issue of finding the global optimal solution in a multi-optimal problem, where the problem contains a number of local and global optimal solutions and the objective is to find the global optimal solution. To compare the power of various algorithms, one of the traditional constrained optimization techniques is compared with both the nontraditional optimization techniques in a multi-optimal problem.

Some algorithms in Chapter 4 use linear programming methods, which are usually taught in operations research and transportation engineering related courses. Sometimes, linear programming methods are also taught in first or second-year undergraduate engineering courses. Thus, a detailed discussion of linear programming methods is avoided in this book. Instead, a brief analysis of the simplex search technique of the linear programming method is given in Appendix A.

The algorithms are presented in a step-by-step format so that they can be easily understood and coded in a computer language. The working principle of each algorithm is also illustrated by showing hand calculations up to a few iterations of the algorithms on a numerical optimization problem. The hand calculations provide a better insight into the working of the optimization algorithms. Moreover, in order to compare the efficiency of different algorithms, as far as possible, the same numerical example is chosen for each algorithm.

Most of the chapters contain at least one working computer code, implementing optimization algorithms presented in the chapter. The computer codes are written in FORTRAN programming language and sample simulation runs obtained under the Microsoft FORTRAN compiler on a PC platform are presented. These codes are also tested with a Unix FORTRAN compiler on a SUN machine. They demonstrate the ease and simplicity with which other optimization algorithms can also be coded. The computer codes presented in the text can be available by sending an e-mail to the author at deb@iitk.ac.in.

The primary objective of this book is to introduce different optimization algorithms to students and design engineers, and provide them with a few computer codes for easy understanding. The mathematical treatment of the algorithms is kept at a less rigorous level so that the text can be used as an introductory book on optimization by design engineers as well as practitioners in industries and by the undergraduate and postgraduate students of engineering institutions. An elementary knowledge of matrix algebra and calculus would be sufficient for understanding most of the materials presented in the book.

Instructors may find this text useful in explaining optimization algorithms and solving numerical examples in the class, although occasional reference to a more theoretical treatment on optimization may be helpful. The best way to utilize this book is to begin with Chapter 1. This chapter helps the reader to correlate the design problems to the optimization problems already discussed. Thereafter, subsequent chapters may be read one at a time. To have a better understanding of the algorithms, the reader must follow the steps of the solved exercise problems as they are outlined in the given algorithm. Then, the progress of each algorithm may be understood by referring to the accompanying figure. For better comprehension, the reader may use the FORTRAN code given at the end of the chapters to solve the example problems. Any comments and suggestions for improving the text would be always welcome.

**Kalyanmoy Deb**



# Acknowledgements

---

The person who introduced me to the field of optimization and who has had a significant role in moulding my career is Professor David E. Goldberg of the University of Illinois at Urbana-Champaign. On a lunch table, he once made me understand that probably the most effective way of communicating one's ideas is through books. That discussion certainly motivated me in taking up this project. The main inspiration for writing this book came from Professor Amitabha Ghosh, Mechanical Engineering Department, IIT Kanpur, when in one tutorial class I showed him the fifty-page handout I prepared for my postgraduate course entitled "Optimization Methods in Engineering Design". Professor Ghosh looked at the handout and encouraged me to revise it in the form of a textbook. Although it took me about an year-and-a-half to execute that revision, I have enjoyed every bit of my experience.

Most of the algorithms presented in this text are collected from various books and research papers related to engineering design optimization. My sincere thanks and appreciation are due to all authors of those books and papers. I have been particularly influenced by the concise and algorithmic approach adopted in the book entitled 'Engineering Optimization—Methods and Applications' by G.V. Reklaitis, A. Ravindran, and K.M. Ragsdell. Many algorithms presented here are modified abstractions from that book.

I am also grateful to Professor Brahma Deo and Dr. Partha Chakroborty for their valuable comments which significantly improved the contents of this book. The computer facility of the Computer Aided Design (CAD) Project, generously provided by Professor Sanjay Dhande, is highly appreciable. My special thanks are due to two of my students N. Srinivas and Ram Bhushan Agrawal for helping me in drawing some of the diagrams and checking some exercise problems. The computer expertise provided by P.V.M. Rao, Samir Kulkarni, and Sailesh Srivastava in preparing one of the computer codes is also appreciated. Discussions with Professors David Blank and M.P. Kapoor on different issues of optimization were also helpful. I am

thankful to my colleagues and staff of the CAD Project for their constant support.

It would have taken at least twice the time to complete this book, if I did not have the privilege to meet Dr. Subhransu Roy who generously provided me with his text-writing and graph plotting softwares. My visits to TELCO, TISCO, Hindustan Motors and Engineers India Ltd, and the discussions I had with many design engineers were valuable in writing some of the chapters. The financial assistance provided by the Continuing Education Centre at the Indian Institute of Technology Kanpur to partially compensate for the preparation of the manuscript is gratefully acknowledged. I also wish to thank the Publishers, PHI Learning for the meticulous care they took in processing the book.

This book could not have been complete without the loving support and encouragement of my wife, Debjani. Her help in typing a significant portion of the manuscript, in proofreading, and in preparing the diagrams has always kept me on schedule. Encouragements from my two children, Debayan and Dhriti, have always motivated me. Finally, I take this opportunity to express my gratitude to my parents, Late Sri Kumud Chandra Deb and Mrs. Chaya Deb, and my loving affection to my brothers—Asis, Debasis, and Subhasis.

**Kalyanmoy Deb**

# 1

## Introduction

---

Optimization algorithms are becoming increasingly popular in multi-engineering design activities, primarily because of the availability and affordability of high speed computers. They are extensively used in those engineering design problems where the emphasis is on maximizing or minimizing a certain goal. For example, optimization algorithms are routinely used in aerospace design activities to minimize the overall weight, simply because every element or component adds to the overall weight of the aircraft. Thus, the minimization of the weight of aircraft components is of major concern to aerospace designers. Chemical engineers, on the other hand, are interested in designing and operating a process plant for an optimum rate of production. Mechanical engineers design mechanical components for the purpose of achieving either a minimum manufacturing cost or a maximum component life. Production engineers are interested in designing optimum schedules of various machining operations to minimize the idle time of machines and the overall job completion time. Civil engineers are involved in designing buildings, bridges, dams, and other structures in order to achieve a minimum overall cost or maximum safety or both. Electrical engineers are interested in designing communication networks so as to achieve minimum time for communication from one node to another.

All the above-mentioned tasks involve either minimization or maximization (collectively known as optimization) of an objective. It is clear from the spectrum of the above problems that it is difficult to discuss the formulation of various engineering optimization problems in a single book. Fortunately, a designer specialized in a particular design is usually more informed about different factors governing that design than anyone else. Thus, as far as the formulation of the optimal problem is concerned, the designer can acquire it with some practice. However, every designer should know a few aspects of the formulation procedure which would help him or her to choose a proper optimization algorithm for the chosen optimal design problem. This requires

a knowledge about the working principles of different optimization methods. In subsequent chapters, we discuss various optimization methods which would hopefully provide some of that knowledge. In this chapter, we demonstrate the optimal problem formulation procedures of four different engineering optimal design problems.

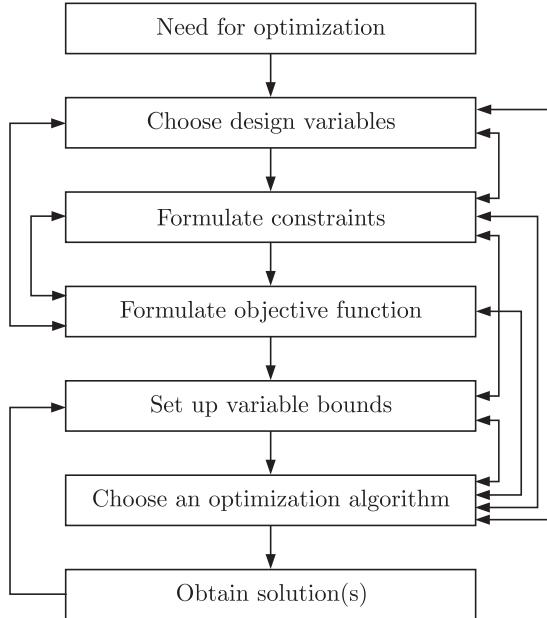
## 1.1 Optimal Problem Formulation

In many industrial design activities, a naive optimal design is achieved by comparing a few (limited up to ten or so) alternative design solutions created by using *a priori* problem knowledge. In such an activity, the feasibility of each design solution is first investigated. Thereafter, an estimate of the underlying objective (cost, profit, etc.) of each design solution is computed and the best design solution is adopted. This naive method is often followed because of the time and resource limitations. But, in many cases this method is followed simply because of the lack of knowledge of the existing optimization procedures. But whatever may be the reason, the purpose of either achieving a quality product or of achieving a competing product is never guaranteed to be fulfilled with the above naive method. Optimization algorithms described in this book provide systematic and efficient ways of creating and comparing new design solutions in order to achieve an optimal design. Since an optimization algorithm requires comparison of a number of design solutions, it is usually time-consuming and computationally expensive. Thus, the optimization procedure must only be used in those problems where there is a definite need of achieving a quality product or a competitive product. It is expected that the design solution obtained through an optimization procedure is better than other solutions in terms of the chosen objective—cost, efficiency, safety, or others.

We begin our discussion with the formulation procedure by mentioning that it is almost impossible to apply a single formulation procedure for all engineering design problems. Since the objective in a design problem and the associated design parameters vary from product to product, different techniques need to be used in different problems. The purpose of the formulation procedure is to create a mathematical model of the optimal design problem, which then can be solved using an optimization algorithm. Since an optimization algorithm accepts an optimization problem in a particular format, every optimal design problem must be formulated in that format. In this section, we discuss different components of that format.

Figure 1.1 shows an outline of the steps usually involved in an optimal design formulation process. As mentioned earlier, the first step is to realize the need for using optimization in a specific design problem. Thereafter, the designer needs to choose the important design variables associated with the design problem. The formulation of optimal design problems involves other considerations, such as constraints, objective function, and variable bounds. As shown in the figure, there is usually a hierarchy in the optimal design

process; although one consideration may get influenced by the other. We discuss all these aspects in the following subsections.



**Figure 1.1** A flowchart of the optimal design procedure.

### 1.1.1 Design Variables

The formulation of an optimization problem begins with identifying the underlying design variables, which are primarily varied during the optimization process. A design problem usually involves many design parameters, of which some are highly sensitive to the proper working of the design. These parameters are called design variables in the parlance of optimization procedures. Other (not so important) design parameters usually remain fixed or vary in relation to the design variables. There is no rigid guideline to choose *a priori* the parameters which may be important in a problem, because one parameter may be more important with respect to minimizing the overall cost of the design, while it may be insignificant with respect to maximizing the life of the product. Thus, the choice of the important parameters in an optimization problem largely depends on the user. However, it is important to understand that the efficiency and speed of optimization algorithms depend, to a large extent, on the number of chosen design variables. In subsequent chapters, we shall discuss certain algorithms which work very efficiently when the number of design variables is small, but do not work that well for a large number of design variables. Thus, by selectively choosing the design variables, the efficacy of the optimization process can be increased. The first *thumb rule* of the formulation of an optimization problem is to choose as

few design variables as possible. The outcome of that optimization procedure may indicate whether to include more design variables in a revised formulation or to replace some previously considered design variables with new design variables.

### 1.1.2 Constraints

Having chosen the design variables, the next task is to identify the constraints associated with the optimization problem. The constraints represent some functional relationships among the design variables and other design parameters satisfying certain physical phenomenon and certain resource limitations. Some of these considerations require that the design remain in static or dynamic equilibrium. In many mechanical and civil engineering problems, the constraints are formulated to satisfy stress and deflection limitations. Often, a component needs to be designed in such a way that it can be placed inside a fixed housing, thereby restricting the size of the component. There is, however, no unique way to formulate a constraint in all problems. The nature and number of constraints to be included in the formulation depend on the user. In many algorithms discussed in this book, it is not necessary to have an explicit mathematical expression of a constraint; but an algorithm or a mechanism to calculate the constraint is mandatory. For example, a mechanical engineering component design problem may involve a constraint to restrain the maximum stress developed anywhere in the component to the strength of the material. In an irregular-shaped component, there may not exist an exact mathematical expression for the maximum stress developed in the component. A finite element simulation software may be necessary to compute the maximum stress. But the simulation procedure and the necessary input to the simulator and the output from the simulator must be understood at this step.

There are usually two types of constraints that emerge from most considerations. Either the constraints are of an inequality type or of an equality type. Inequality constraints state that the functional relationships among design variables are either greater than, smaller than, or equal to, a resource value. For example, the stress ( $\sigma(x)$ ) developed anywhere in a component must be smaller than or equal to the allowable strength ( $S_{\text{allowable}}$ ) of the material. Mathematically,

$$\sigma(x) \leq S_{\text{allowable}}.$$

Most of the constraints encountered in engineering design problems are of this type. Some constraints may be of greater-than-equal-to type: for example, the natural frequency ( $\nu(x)$ ) of a system may required to be greater than 2 Hz, or mathematically,  $\nu(x) \geq 2$ . Fortunately, one type of inequality constraints can be transformed into the other type by multiplying both sides by  $-1$  or by interchanging the left and right sides. For example, the former constraint can be transformed into a greater-than-equal-to type by either  $-\sigma(x) \geq -S_{\text{allowable}}$  or  $S_{\text{allowable}} \geq \sigma(x)$ .

Equality constraints state that the functional relationships should exactly match a resource value. For example, a constraint may require that the deflection ( $\delta(x)$ ) of a point in the component must be exactly equal to 5 mm, or mathematically,

$$\delta(x) = 5.$$

Equality constraints are usually more difficult to handle and, therefore, need to be avoided wherever possible. If the functional relationships of equality constraints are simpler, it may be possible to reduce the number of design variables by using the equality constraints. In such a case, the equality constraints reduce the complexity of the problem, thereby making it easier for the optimization algorithms to solve the problem. In Chapter 4, we discuss a number of algorithms which are specially designed to handle equality constraints. Fortunately, in many engineering design optimization problems, it may be possible to relax an equality constraint by including two inequality constraints. The above deflection equality constraint can be replaced by two constraints:

$$\delta(x) \geq 4,$$

$$\delta(x) \leq 6.$$

The exact deflection requirement of 5 mm is relaxed by allowing it to deflect anywhere between 4 mm to 6 mm. Although this formulation is inexact as far as the original requirement is concerned, this flexibility allows a smoother operation of the optimization algorithms. Thus, the second *thumb* rule in the formulation of optimization problems is that the number of complex equality constraints should be kept as low as possible.

### 1.1.3 Objective Function

The third task in the formulation procedure is to find the objective function in terms of the design variables and other problem parameters. The common engineering objectives involve minimization of overall cost of manufacturing, or minimization of overall weight of a component, or maximization of net profit earned, or maximization total life of a product, or others. Although most of the above objectives can be quantified (expressed in a mathematical form), there are some objectives that may not be quantified easily. For example, the esthetic aspect of a design, ride characteristics of a car suspension design, and reliability of a design are important objectives that one may be interested in maximizing in a design, but the exact mathematical formulation may not be possible. In such a case, usually an approximating mathematical expression is used. Moreover, in any real-world optimization problem, there could be more than one objective that the designer may want to optimize simultaneously. Even though a few multi-objective optimization algorithms exist in the literature (Chankong and Haimes, 1983), they are complex and computationally expensive. Thus, in most optimal design problem, multiple objectives are avoided. Instead, the designer chooses the most important

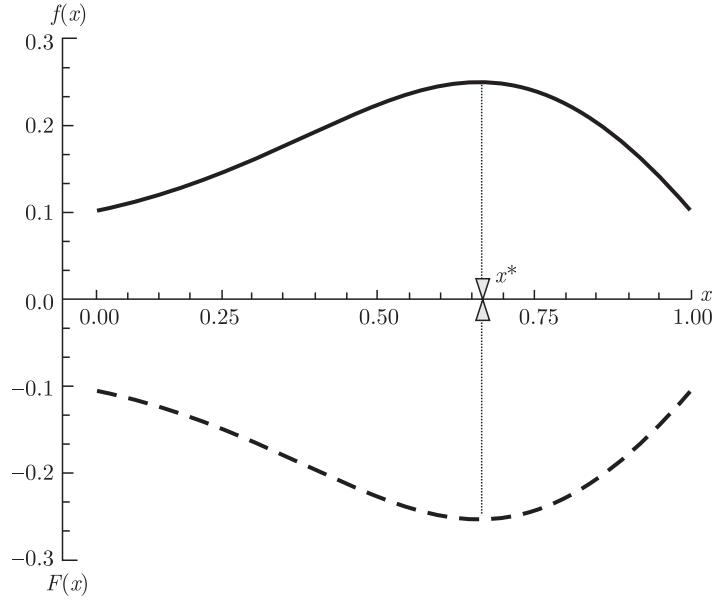
objective as the objective function of the optimization problem, and the other objectives are included as constraints by restricting their values within a certain range. For example, consider an optimal truss structure design problem. The designer may be interested in minimizing the overall weight of the structure and simultaneously be concerned in minimizing the deflection of a specific point in the truss. In the optimal problem formulation, the designer may like to use the weight of the truss (as a function of the cross-sections of the members) as the objective function and have a constraint with the deflection of the concerned point to be less than a specified limit. In general, the objective function is not required to be expressed in a mathematical form. A simulation package may be required to evaluate the objective function. But whatever may be the way to evaluate the objective function, it must be clearly understood.

The objective function can be of two types. Either the objective function is to be maximized or it has to be minimized. Unfortunately, the optimization algorithms are usually written either for minimization problems or for maximization problems. Although in some algorithms, some minor structural changes would enable to perform either minimization or maximization, this requires extensive knowledge of the algorithm. Moreover, if an optimization software is used for the simulation, the modified software needs to be compiled before it can be used for the simulation. Fortunately, the *duality* principle helps by allowing the same algorithm to be used for minimization or maximization with a minor change in the objective function instead of a change in the entire algorithm. If the algorithm is developed for solving a minimization problem, it can also be used to solve a maximization problem by simply multiplying the objective function by  $-1$  and vice versa. For example, consider the maximization of the single-variable function  $f(x) = x^2(1 - x)$  shown by a solid line in Figure 1.2. The maximum point happens to be at  $x^* = 0.667$ . The duality principle suggests that the above problem is equivalent to minimizing the function  $F(x) = -x^2(1 - x)$ , which is shown by a dashed line in Figure 1.2. The figure shows that the minimum point of the function  $F(x)$  is also at  $x^* = 0.667$ . Thus, the optimum solution remains the same. But once we obtain the optimum solution by minimizing the function  $F(x)$ , we need to calculate the optimal function value of the original function  $f(x)$  by multiplying  $F(x)$  by  $-1$ .

#### 1.1.4 Variable Bounds

The final task of the formulation procedure is to set the minimum and the maximum bounds on each design variable. Certain optimization algorithms do not require this information. In these problems, the constraints completely surround the feasible region. Other problems require this information in order to confine the search algorithm within these bounds. In general, all  $N$  design variables are restricted to lie within the minimum and the maximum bounds as follows:

$$x_i^{(L)} \leq x_i \leq x_i^{(U)} \quad \text{for } i = 1, 2, \dots, N.$$



**Figure 1.2** Illustration of the duality principle. The maximum point of  $f(x)$  is the same as the minimum point of  $F(x)$ .

In any given problem, the determination of the variables bounds  $x_i^{(L)}$  and  $x_i^{(U)}$  may be difficult. One way to remedy this situation is to make a guess about the optimal solution and set the minimum and maximum bounds so that the optimal solution lies within these two bounds. After simulating the optimization algorithm once, if the optimal solution is found to lie within the chosen variable bounds, there is no problem. On the other hand, if any design variable corresponding to the optimal solution is found to lie on or near the minimum or the maximum bound, the chosen bound may not be correct. The chosen bound may be readjusted and the optimization algorithm may be simulated again. Although this strategy may seem to work only with linear problems, it has been found useful in many real-world engineering optimization problems.

After the above four tasks are completed, the optimization problem can be mathematically written in a special format, known as *nonlinear programming* (NLP) format. Denoting the design variables as a column vector<sup>1</sup>  $x = (x_1, x_2, \dots, x_N)^T$ , the objective function as a scalar quantity  $f(x)$ ,  $J$  inequality constraints as  $g_j(x) \geq 0$ , and  $K$  equality constraints as

$$h_k(x) = 0$$

---

<sup>1</sup>The representation of the design variables in the above column vector helps to achieve some matrix operations in certain multivariable optimization methods described in Chapters 3 and 4.

we write the NLP problem:

$$\left. \begin{array}{l}
 \text{Minimize} \quad f(x) \\
 \text{subject to} \\
 g_j(x) \geq 0, \quad j = 1, 2, \dots, J; \\
 h_k(x) = 0, \quad k = 1, 2, \dots, K; \\
 x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \dots, N.
 \end{array} \right\} \quad (1.1)$$

Note that the above formulation can represent a formulation for maximization problems by using the duality principle and can represent a formulation for problems with the lesser-than-equal-to type inequality constraints by using the techniques described earlier. However, the optimization algorithm used to solve the above NLP problem depends on the type of the objective function and constraints. It is important to note that the constraints must be written in a way so that the right-side of the inequality or equality sign is zero.

It is worth mentioning here that all the above four tasks are not independent of each other. While formulating the objective function, the designer may decide to include or delete some constraints. In many problems, while the constraints are being formulated, it is necessary to add some artificial design variables, which make the overall formulation easier. The update of the design variables, the constraints, the objective function, and the variable bounds may continue for a few iterations until the designer is finally satisfied with a reasonable formulation. Certain possible iterations are shown in Figure 1.1. We may mention here that this update also depends on the knowledge of the optimization algorithms to be used to solve the problem. But this requires some practice of using optimization algorithms before such input may be incorporated into the formulation procedure. Nevertheless, after the optimization problem is formulated, an optimization algorithm is chosen and an optimal solution of the NLP problem is obtained. We now illustrate the above four steps of the formulation procedure in four engineering optimal design problems.

## 1.2 Engineering Optimization Problems

Optimization problems can be found in most engineering disciplines. Because of the variety of engineering design problems, it is not possible to discuss the formulation of every optimization problem that is encountered in engineering design in a single book. However, in this section, we categorize different engineering problem-solving tasks in which an optimization method can be applied to find a solution.

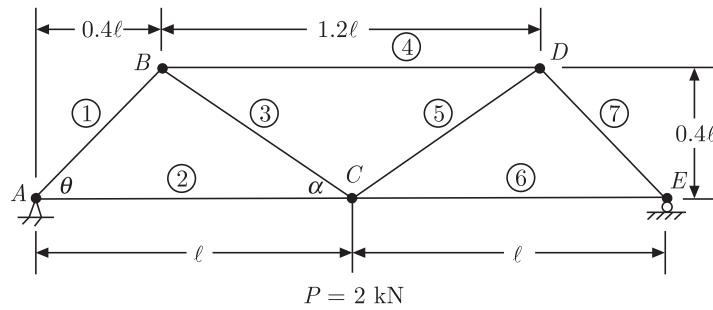
### 1.2.1 Design and Manufacturing

A majority of the optimization problems arise from design and manufacturing areas. Since designs have to work efficiently, have to be cost-effective, have to have least weight in certain applications, and have to satisfy many other criteria, it makes perfect sense to apply an optimization algorithm to find such a solution. Similarly, a manufacturing process involves various parameters and operations that must be optimized to produce a competitive product or to achieve a quality product. Hence, the use of an optimization procedure is essential in most manufacturing problem-solving tasks. To illustrate, here we only discuss two design problem-solving tasks and one manufacturing process optimization task.

#### Optimal design of a truss structure

A truss structure is used in many civil engineering applications including bridges, buildings, and roofs. There are two different types of optimization problems in a truss structure design. Firstly, the topology of the truss structure (the connectivity of the elements in a truss) could be an optimization problem. In this problem, the objective is to find the optimal connectivity of truss elements so as to achieve the minimal cost of materials and construction. Secondly, once the optimal layout of the truss is known, the determination of every element cross-section is another optimization problem. In this problem, the objective is to find the optimal cross-sections of all elements in order to achieve a minimum cost of materials and construction. Although both these problems attempt to achieve the same objective, the search space and the optimization algorithm required to solve each problem are different. Here, we discuss the latter problem only. However, there exist certain algorithms which can be used to solve both the above problems simultaneously. We discuss more about these algorithms in Chapter 6.

Consider the seven-bar truss structure shown in Figure 1.3. The loading is also shown in the figure. The length of the members  $AC = CE = \ell = 1$  m. Once the connectivity of the truss is given, the cross-sectional area and the material properties of the members are the design parameters. Let us choose the cross-sectional area of members as the design variables for this problem. There are seven design variables, each specifying the cross-section of a member



**Figure 1.3** A typical seven-bar truss structure.

( $A_1$  to  $A_7$ ). Using the symmetry of the truss structure and loading, we observe that for the optimal solution,  $A_7 = A_1$ ,  $A_6 = A_2$ , and  $A_5 = A_3$ . Thus, there are practically four design variables ( $A_1$  to  $A_4$ ). This completes the first task of the optimization procedure.

The next task is to formulate the constraints. In order for the truss to carry the given load  $P = 2$  kN, the tensile and compressive stress generated in each member must not be more than the corresponding allowable strength  $S_{yt}$  and  $S_{yc}$  of the material. Let us assume that the material strength for all elements is  $S_{yt} = S_{yc} = 500$  MPa and the modulus of elasticity  $E = 200$  GPa. For the given load, we can compute the axial force generated in each element (Table 1.1). The positive force signifies tensile load and the negative force signifies compressive load acting on the member.

**Table 1.1** Axial Force in Each Member of the Truss

Member	Force	Member	Force
$AB$	$-\frac{P}{2} \csc \theta$	$BC$	$+\frac{P}{2} \csc \alpha$
$AC$	$+\frac{P}{2} \cot \theta$	$BD$	$-\frac{P}{2} (\cot \theta + \cot \alpha)$

Thereafter, the axial stress can be calculated by dividing the axial load by the cross-sectional area of that member. Thus, the first set of constraints can be written as

$$\frac{P \csc \theta}{2A_1} \leq S_{yc},$$

$$\frac{P \cot \theta}{2A_2} \leq S_{yt},$$

$$\frac{P \csc \alpha}{2A_3} \leq S_{yt},$$

$$\frac{P}{2A_4} (\cot \theta + \cot \alpha) \leq S_{yc}.$$

In the above truss structure,  $\tan \theta = 1.0$  and  $\tan \alpha = 2/3$ . The other set of constraints arises from the stability consideration of the compression members  $AB$ ,  $BD$ , and  $DE$ . Realizing that each of these members is connected by pin joints, we can write the Euler buckling conditions for the axial load in members  $AB$  and  $BD$  (Shigley, 1986):

$$\frac{P}{2 \sin \theta} \leq \frac{\pi E A_1^2}{1.281 \ell^2},$$

$$\frac{P}{2} (\cot \theta + \cot \alpha) \leq \frac{\pi E A_4^2}{5.76 \ell^2}.$$

In most structures, deflection is a major consideration. In the above truss structure, let us assume that the maximum vertical deflection at  $C$  is  $\delta_{\max} = 2$  mm. By using Castiglano's theorem (Timoshenko, 1986), we obtain the deflection constraint:

$$\frac{P\ell}{E} \left( \frac{0.566}{A_1} + \frac{0.500}{A_2} + \frac{2.236}{A_3} + \frac{2.700}{A_4} \right) \leq \delta_{\max}.$$

All the above constraints are of less-than-equal-to type. Once the constraints are formulated, the next task is to formulate the objective function. In this problem, we are interested in minimizing the weight of the truss structure. Since we have assumed the same material for all members, the minimization of the total volume of material will yield the same optimal solution as the minimization of the total weight. Thus, we write the objective function as

$$\text{Minimize } 1.132A_1\ell + 2A_2\ell + 1.789A_3\ell + 1.2A_4\ell.$$

The fourth task is to set some lower and upper bounds for the four cross-sectional areas. We may choose to make all four areas lie between 10 and 500 mm<sup>2</sup>. Thus, the variable bounds are as

$$10 \times 10^{-6} \leq A_1, A_2, A_3, A_4 \leq 500 \times 10^{-6}.$$

In the following, we present the above truss structure problem in NLP form, which is suitable for solving by using an optimization algorithm described in Chapter 4:

$$\text{Minimize } 1.132A_1\ell + 2A_2\ell + 1.789A_3\ell + 1.2A_4\ell$$

subject to

$$S_{yc} - \frac{P}{2A_1 \sin \theta} \geq 0,$$

$$S_{yt} - \frac{P}{2A_2 \cot \theta} \geq 0,$$

$$S_{yt} - \frac{P}{2A_3 \sin \alpha} \geq 0,$$

$$S_{yc} - \frac{P}{2A_4} (\cot \theta + \cot \alpha) \geq 0,$$

$$\frac{\pi EA_1^2}{1.281\ell^2} - \frac{P}{2 \sin \theta} \geq 0,$$

$$\frac{\pi EA_4^2}{5.76\ell^2} - \frac{P}{2} (\cot \theta + \cot \alpha) \geq 0,$$

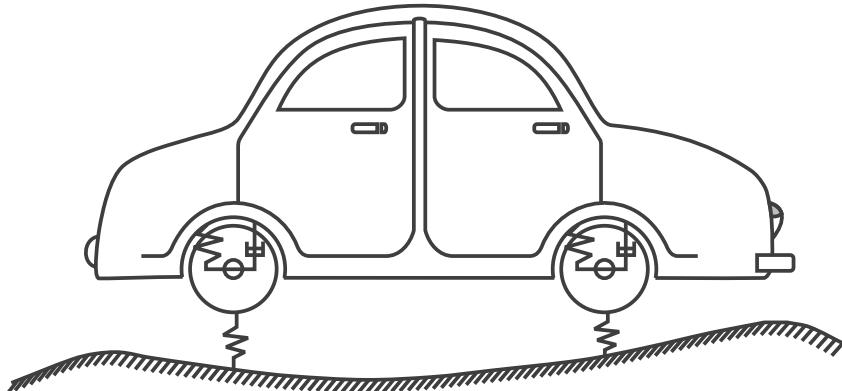
$$\delta_{\max} - \frac{P\ell}{E} \left( \frac{0.566}{A_1} + \frac{0.500}{A_2} + \frac{2.236}{A_3} + \frac{2.700}{A_4} \right) \geq 0,$$

$$10 \times 10^{-6} \leq A_1, A_2, A_3, A_4 \leq 500 \times 10^{-6}.$$

This shows the formulation of the truss structure problem (Deb et al., 2000). The seven-bar truss shown in Figure 1.3 is statically determinate and the axial force, stress, and deflection were possible to compute exactly. In cases where the truss is statically indeterminate and large (for hand calculations), the exact computations of stress and deflection may not be possible. A finite element software may be necessary to compute the stress and deflection in any member and at any point in the truss. Although similar constraints can then be formulated with the simulated stresses and deflections, the optimization algorithm which may be used to solve the above seven-bar truss problem may not be efficient to solve the resulting NLP problem for statically indeterminate or large truss problems. The difficulty arises due to the inability to compute the gradients of the constraints. We shall discuss more about this aspect in Chapter 4.

### Optimal design of a car suspension

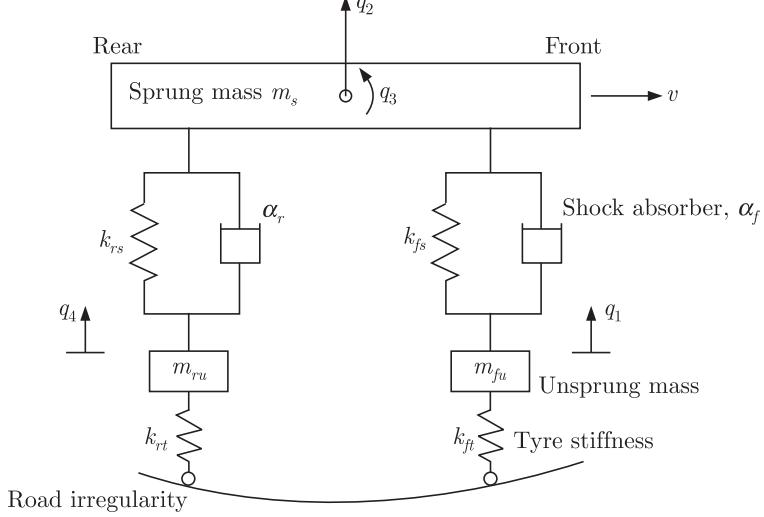
The comfort in riding a car largely depends on the suspension characteristics. The car body is usually supported by a suspension coil spring and a damper at each wheel (Figure 1.4).



**Figure 1.4** A two-dimensional model of a car suspension system.

In some cars, the axle assembly is directly supported on the wheel. The tyre of the wheel can also be assumed to have some stiffness in the vertical direction. A two-dimensional dynamic model of a car suspension system is shown in Figure 1.5. In this model, only two wheels (one each at rear and front) are considered. The sprung mass of the car is considered to be supported on two axles (front and rear) by means of a suspension coil spring and a shock absorber (damper). Each axle contains some unsprung mass which is supported by the tyre.

In order to formulate the optimal design problem, the first task is to identify the important design variables. Let us first identify all the design parameters that could govern the dynamic behaviour of the car vibration. In the following, we list all these parameters:



**Figure 1.5** The dynamic model of the car suspension system. The above model has four degrees-of-freedom ( $q_1$  to  $q_4$ ).

Sprung mass  $m_s$ ,

Front coil stiffness  $k_{fs}$ ,

Front unsprung mass  $m_{fu}$ ,

Rear coil stiffness  $k_{rs}$ ,

Rear unsprung mass  $m_{ru}$ ,

Front tyre stiffness  $k_{ft}$ ,

Rear damper coefficient  $\alpha_r$ ,

Rear tyre stiffness  $k_{rt}$ ,

Front damper coefficient  $\alpha_f$ ,

Axle-to-axle distance  $\ell$ ,

Polar moment of inertia of the car  $J$ .

We may consider all the above parameters as design variables, but the time taken for convergence of the optimization algorithm may be too much. In order to simplify the formulation, we consider only four of the above parameters—front coil stiffness  $k_{fs}$ , rear coil stiffness  $k_{rs}$ , front damper coefficient  $\alpha_f$ , and rear damper coefficient  $\alpha_r$ —as design variables. We keep the other design parameters as constant:

$$\begin{aligned} m_s &= 1000 \text{ kg}, & m_{fu} &= 70 \text{ kg}, & m_{ru} &= 150 \text{ kg}, \\ k_{ft} &= 20 \text{ kg/mm}, & k_{rt} &= 20 \text{ kg/mm}, & J &= 550 \text{ kg-m}^2, \\ \ell_1 &= 1.6 \text{ m}, & \ell_2 &= 1.6 \text{ m}, & \ell &= 3.2 \text{ m}. \end{aligned}$$

The parameters  $\ell_1$  and  $\ell_2$  are the horizontal distance of the front and rear axles from the centre of gravity of the sprung mass. Using these parameters, differential equations governing the vertical motion of the unsprung mass at the front axle ( $q_1$ ), the sprung mass ( $q_2$ ), and the unsprung mass at the

rear axle ( $q_4$ ), and the angular motion of the sprung mass ( $q_3$ ) are written (Figure 1.5):

$$\ddot{q}_1 = (F_2 + F_3 - F_1)/m_{fu}, \quad (1.2)$$

$$\ddot{q}_2 = -(F_2 + F_3 + F_4 + F_5)/m_s, \quad (1.3)$$

$$\ddot{q}_3 = [(F_4 + F_5)\ell_2 - (F_2 + F_3)\ell_1]/J, \quad (1.4)$$

$$\ddot{q}_4 = (F_4 + F_5 - F_6)/m_{ru}, \quad (1.5)$$

where the forces  $F_1$  to  $F_6$  are calculated as follows:

$$\begin{aligned} F_1 &= k_{ft}d_1, & F_2 &= k_{fs}d_2, & F_3 &= \alpha_f \dot{d}_2, \\ F_4 &= k_{rs}d_4, & F_5 &= \alpha_r \dot{d}_4, & F_6 &= k_{rt}d_3. \end{aligned}$$

The parameters  $d_1$ ,  $d_2$ ,  $d_3$ , and  $d_4$  are the relative deformations in the front tyre, the front spring, the rear tyre, and the rear spring, respectively. Figure 1.5 shows all the four degrees-of-freedom of the above system ( $q_1$  to  $q_4$ ). The relative deformations in springs and tyres can be written as follows:

$$\begin{aligned} d_1 &= q_1 - f_1(t), \\ d_2 &= q_2 + \ell_1 q_3 - q_1, \\ d_3 &= q_4 - f_2(t), \\ d_4 &= q_2 - \ell_2 q_3 - q_4. \end{aligned}$$

The time-varying functions  $f_1(t)$  and  $f_2(t)$  are road irregularities as functions of time. Any function can be used for  $f_1(t)$ . For example, a bump can be modelled as  $f_1(t) = A \sin(\pi t/T)$ , where  $A$  is the amplitude of the bump and  $T$  the time required to cross the bump. When a car is moving forward, the front wheel experiences the bump first, while the rear wheel experiences the same bump a little later, depending upon the speed of the car. Thus, the function  $f_2(t)$  can be written as  $f_2(t) = f_1(t - \ell/v)$ , where  $\ell$  is the axle-to-axle distance and  $v$  is the speed of the car. For the above bump,  $f_2(t) = A \sin(\pi(t - \ell/v)/T)$ . The coupled differential equations specified in Equations (1.2) to (1.5) can be solved using a numerical integration technique (for example, a fourth-order Runge-Kutta method can be used) to obtain the pitching and bouncing dynamics of the sprung mass  $m_s$ . Equations can be integrated for a time range from zero to  $t_{\max}$ .

After the design variables are chosen, the next task is to formulate the constraints associated with the above car suspension problem. In order to simplify the problem, we consider only one constraint. The *jerk* (the rate of change of the vertical acceleration of the sprung mass) is a major factor concerning the comfort of the riding passengers. The guideline used in car

industries suggests that the maximum jerk experienced by the passengers should not be more than about 18 m/s<sup>3</sup>. Mathematically,

$$\max q_2'''(t) \leq 18.$$

When the four coupled differential equations (1.2) to (1.5) are solved, the above constraint can be computed by numerically differentiating the vertical movement of the sprung mass ( $q_2$ ) thrice with respect to time.

The next task is to formulate the objective function. In this problem, the primary objective is to minimize the transmissibility factor which is calculated as the ratio of the bouncing amplitude  $q_2(t)$  of the sprung mass to the road excitation amplitude  $A$ . Thus, we write the objective function as

$$\text{Minimize } \frac{\max \text{ abs } q_2(t)}{A}.$$

The above objective function can be calculated from the solution of the four differential equations mentioned earlier. A minimum value of the transmissibility factor suggests the minimum transmission of road vibration to the passengers. This factor is also directly related to the *ride* characteristics as specified by the ISO standard. Thus, the optimized design of the above car suspension system would provide the minimum transmissibility of the road vibration to the passengers with a limited level of jerk.

Finally, a minimum and maximum limit for each design variable can be set. This may require some previous experience with a car suspension design, but the following limits for the above car may include the optimal solution:

$$0 \leq k_{fs}, k_{rs} \leq 2 \text{ kg/mm},$$

$$0 \leq \alpha_f, \alpha_r \leq 300 \text{ kg/(m/s)}.$$

Thus, the above optimal car suspension design problem can be written in NLP form as follows (Deb and Saxena, 1997):

$$\text{Minimize } \frac{\max \text{ abs } q_2(t)}{A}$$

subject to

$$18 - \max q_2'''(t) \geq 0,$$

$$0 \leq k_{fs}, k_{rs} \leq 2,$$

$$0 \leq \alpha_f, \alpha_r \leq 300.$$

### Metal cutting problem

A steel bar is to be machined on a CNC lathe, using P20 carbide tool (Sardiñas et al., 2006). The lathe has a 10 kW motor and a transmission efficiency of 75%. A maximum allowable cutting force of 5,000 N is applied. It is

considered that the operation will remove 219,912 mm<sup>3</sup> of material. The set-up time, tool-change time and the time during which the tool does not cut have been assumed as 0.15, 0.20 and 0.05 min, respectively. The objectives are minimization of operation time ( $T_p$ ) and used tool life ( $\xi$ ). The multi-objective optimization problem formulation is given below (Deb and Datta, 2012):

$$\text{Minimize } F_1(\mathbf{x}) = T_p(\mathbf{x}),$$

$$\text{Minimize } F_2(\mathbf{x}) = \xi(\mathbf{x}),$$

subject to

$$g_1(\mathbf{x}) \equiv 1 - \frac{P(\mathbf{x})}{\eta P^{\max}} \geq 0,$$

$$g_2(\mathbf{x}) \equiv 1 - \frac{F_c(\mathbf{x})}{F_c^{\max}} \geq 0,$$

$$g_3(\mathbf{x}) \equiv 1 - \frac{R(\mathbf{x})}{R^{\max}} \geq 0,$$

$$x_i^{\min} \leq x_i \leq x_i^{\max},$$

where

$$T_p(\mathbf{x}) = 0.15 + 219,912 \left( \frac{1 + \frac{0.20}{T(\mathbf{x})}}{MRR(\mathbf{x})} \right) + 0.05,$$

$$\xi(\mathbf{x}) = \frac{219,912}{MRR(\mathbf{x}) T(\mathbf{x})} \times 100,$$

$$T(\mathbf{x}) = \frac{5.48(10^9)}{v^{3.46} f^{0.696} a^{0.460}},$$

$$F_c(\mathbf{x}) = \frac{6.56(10^3) f^{0.917} a^{0.1.10}}{v^{0.286}},$$

$$P(\mathbf{x}) = \frac{v F_c}{60,000}, \quad MRR(\mathbf{x}) = 1,000 v f a,$$

$$R(\mathbf{x}) = \frac{125 f^2}{r_n}.$$

The objective  $\xi(\mathbf{x})$  is considered as the part of the whole tool life which is consumed in the machining process, hence an operating condition that will minimize this objective will use the machining task optimally. Here,  $R$  is the surface roughness and  $r_n$  is the nose radius of the tool. The constant parameter values are given below:

$$P^{\max} = 10 \text{ (kW)}, \quad F_c^{\max} = 5,000 \text{ N}, \quad R^{\max} = 50 \text{ } \mu\text{m},$$

$$r_n = 0.8 \text{ mm}, \quad \eta = 0.75.$$

Following variable bounds can be used for cutting speed, feed, and depth of cut:

$$\begin{aligned} v_{\min} &= 250 \text{ m/min}, & v_{\max} &= 400 \text{ m/min}, \\ f_{\min} &= 0.15 \text{ mm/rev}, & f_{\max} &= 0.55 \text{ mm/rev}, \\ a_{\min} &= 0.5 \text{ mm}, & a_{\max} &= 6 \text{ mm}. \end{aligned}$$

### 1.2.2 Modelling

The next major application area of optimization methods is in modelling of systems and processes that are often faced in engineering studies. Modelling refers to a task that is used to describe a system or process in a manner that can be later analyzed to gather further information about the system or process. Often a crisp and clear idea about the working principle in any reasonable scientific detail is usually not known for most engineering systems. Take, for example, modelling a blast furnace in which steel is produced from raw materials, such as iron ore, sinter, coke, etc. Given all the input materials and their sizes and compositions that are fed into a blast furnace, the quality of the produced steel after some hours of its operation cannot be written in mathematical form in any accurate way. This is because the quality of the raw materials is not uniform across their charges. Moreover, all heat and mass transfer phenomena and chemical reactions may not be known appropriately. In such a scenario, a mathematical modelling of such a system becomes a difficult task. However, steel has been produced for more than 100 years and there exists a plethora of data (input raw material information versus output quality of steel production) with most steel plants. In such a scenario, there are two types of modelling tasks that can be performed:

- (i) **Manipulation of governing equations:** Whatever scientific knowledge is available about the system or process, they can be used by using some modification parameters. For example, if the following system of (hypothetical) partial differential equations (PDEs) describes a particular process:

$$\begin{aligned} \frac{\partial E}{\partial x} &= a_{11}x^2 + a_{12}y, \\ \frac{\partial E}{\partial y} &= a_{21}xy^2 + a_{12}y^3, \end{aligned}$$

then the difference between the predicted  $E$  values (obtained by solving above equations) and the actual  $E$  values (obtained from practice) can be taken care of by modifying the above PDEs as follows:

$$\begin{aligned} \frac{\partial E}{\partial x} &= \beta_{11}a_{11}x^2 + \beta_{12}a_{12}y, \\ \frac{\partial E}{\partial y} &= \beta_{21}a_{21}xy^2 + \beta_{22}a_{12}y^3. \end{aligned}$$

In the above PDEs, the  $\beta$ -terms are unknown. We can then formulate an optimization procedure to find a set of values of  $\beta$ -terms so that the difference between simulated and observed  $E$  values is minimum:

$$\text{Minimize } f(\boldsymbol{\beta}) = (E^{\text{observed}} - E^{\text{simulated}})^2,$$

subject to  $E^{\text{simulated}} = \text{Solution (PDEs)}$ .

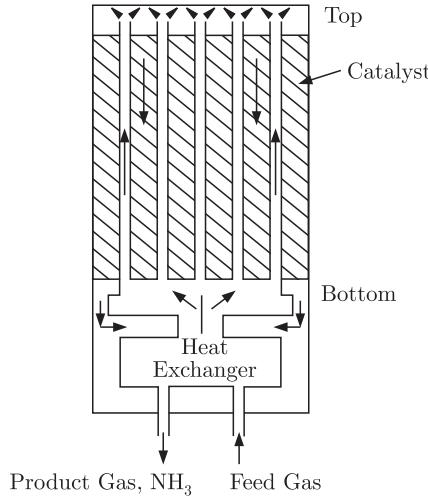
For a given point  $\boldsymbol{\beta}$  in the search space, the above PDEs can be numerically (or exactly, if possible) solved and simulated  $E$  values can be found as a function of  $x$  and  $y$ . These simulated values can then be compared with observed values from practice and the objective function  $f(\boldsymbol{\beta})$  can be calculated. The optimization algorithm should drive the search towards a solution  $\boldsymbol{\beta}^*$  that would correspond to the minimum possible difference. The advantage with such a procedure is that at the end a mathematical expression of  $E(x, y)$  can be known and a further analysis can be performed to optimize its performance and get useful insights about an optimal operation of a system or a plant.

- (ii) **Use of non-mathematical techniques:** Most recent modelling tasks are performed using a non-mathematical procedure in which instead of arriving at a mathematical function of the model, the modelling information is stored in a network of entities or by some other means. One such common approach is to use artificial neural networks (ANN) (Haykin, 1999). In an ANN, a network of connectivities between input and output layers through a series of hidden layers is established. This is done by first choosing a network of connectivities and then employing an ANN learning algorithm to arrive at optimal connection weight for each connection link by minimizing the error between ANN and observed output data. This process is called the training phase of ANN. One popular approach is the use of back-propagation learning algorithm (Rumelhart and McClelland, 1986). The learning optimization task is achieved through an optimization procedure. Both the tasks of finding the optimal connectivity and the optimal connection weights can be achieved by other more sophisticated ANN methodologies. However, at the end of the ANN learning process, the optimal network is capable of providing output value of a new input set accurately, provided the new test set interpolates the training data well. Thus, such an ANN model can be used to perform an analysis of the system or process for a better understanding of the system.

Here, we illustrate a mathematical modelling task as an optimization problem.

### Optimal design of an ammonia reactor

In an ammonia reactor design problem, feed gas containing nitrogen, hydrogen, methane, argon, and a small percentage of ammonia enters the bottom of the reactor (Figure 1.6).



**Figure 1.6** A typical ammonia synthesis reactor.

Thereafter, the feed gas rises till it reaches the top of the reactor. Then, while moving downward, the nitrogen and hydrogen present in the feed gas undergo reaction to form ammonia in the presence of a catalyst placed in the reactor. The production of ammonia depends on the temperature of the feed gas, the temperature at the top of the reactor, the partial pressures of the reactants (nitrogen and hydrogen), and the reactor length. The modelling task attempts to arrive at a mathematical governing reaction equations so as to correctly predict certain problem parameters.

In this problem, for a given reactor length  $x$ , we identify three design parameters—the molar flow rate of nitrogen per unit catalyst area  $N_{N_2}$ , the feed gas temperature  $T_f$ , and the reacting gas temperature  $T_g$ . In order to maintain the energy balance of reactions in the reactor, three coupled differential equations must be satisfied (Murase, Roberts, and Converse, 1970; Upadhyay and Deb, 1997). First, the decrease in the feed gas temperature must be according to the heat loss to the reaction gas:

$$\frac{dT_f}{dx} = -\frac{US_1}{WC_{pf}}(T_g - T_f). \quad (1.6)$$

In Equation (1.6),  $U$  is the overall heat transfer coefficient,  $S_1$  is the surface area of the catalyst tubes per unit reactor length,  $W$  is the total mass flow rate, and  $C_{pf}$  is the specific heat capacity of the feed gas. Secondly, the change in the reaction gas temperature must be according to the heat gain from the feed gas and heat generated in the reaction:

$$\begin{aligned} \frac{dT_g}{dx} &= -\frac{US_1}{WC_{pg}}(T_g - T_f) + \frac{(-\Delta H)S_2}{WC_{pg}} f_a \\ &\times \left( K_1 \frac{1.5p_{N_2}p_{H_2}}{p_{NH_3}} - K_2 \frac{p_{NH_3}}{1.5p_{H_2}} \right), \end{aligned} \quad (1.7)$$

where the parameters  $K_1$  and  $K_2$  are as follows:

$$K_1 = 1.78954 \times 10^4 \exp[-20800/(RT_g)],$$

$$K_2 = 2.5714 \times 10^{16} \exp[-47400/(RT_g)].$$

The parameter  $S_2$  denotes the cross-sectional area of the catalyst zone,  $\Delta H$  is the heat of reaction,  $C_{pg}$  is the specific heat capacity of the reacting gas,  $f_a$  is the catalyst activity, and  $R$  is the ideal gas constant. The parameters  $p_{N_2}$ ,  $p_{H_2}$ , and  $p_{NH_3}$  are the partial pressures of nitrogen, hydrogen, and ammonia, respectively. Thirdly, the mass balance of nitrogen yields

$$\frac{dN_{N_2}}{dx} = -f_a \left( K_1 \frac{1.5p_{N_2}p_{H_2}}{p_{NH_3}} - K_2 \frac{p_{NH_3}}{1.5p_{H_2}} \right). \quad (1.8)$$

The partial pressures in the above differential equations are computed as

$$p_{N_2} = \frac{286N_{N_2}}{2.598N_{N_2} + 2N_{N_2}},$$

$$p_{H_2} = 3p_{N_2},$$

$$p_{NH_3} = \frac{286(2.23N_{N_2} - 2N_{N_2})}{2.598N_{N_2} + 2N_{N_2}},$$

where  $N_{N_2}$  is the molar flow rate of nitrogen per unit catalyst area at the top of the reactor. We use the following parameter values:

$$C_{pf} = 0.707 \text{ kcal}/(\text{kg K}), \quad S_1 = 10 \text{ m},$$

$$C_{pg} = 0.719 \text{ kcal}/(\text{kg K}), \quad S_2 = 0.78 \text{ m},$$

$$\Delta H = -26,600 \text{ kcal/kmol nitrogen}, \quad R = 1.987 \text{ kcal}/(\text{kmol K}),$$

$$U = 500 \text{ kcal}/(\text{m}^2 \text{ h K}), \quad W = 26,400 \text{ kg/h},$$

$$f_a = 1.$$

Note that all the above three ordinary differential equations (ODEs) (1.6)–(1.8) are coupled to each other. In order to solve these equations, we use the following boundary conditions:

$$T_f(x = 0) = T_0,$$

$$T_g(x = 0) = 694 \text{ K},$$

$$N_{N_2}(x = 0) = 701.2 \text{ kmol}/(\text{m}^2 \text{ h}).$$

The three constraints (Equations (1.6) to (1.8)) can be used to compute the three parameters of the problem. By comparing these three parameters with

their recorded values from an actual reactor, the chemical process can be modelled. There are various practical reasons why the solutions of the above ODEs usually do not match with the plant values. By using the objective of the model optimization problem as minimization of the difference between simulated parameter values and their actual plant values, we hope to represent the model of the reactor with the ODEs. Thus, the NLP problem is as follows:

$$\text{Minimize} \quad \left( N_{\text{N}_2} - N_{\text{N}_2}^{\text{plant}} \right)^2 + (T_f - T_f^{\text{plant}})^2 + (T_g - T_g^{\text{plant}})^2,$$

subject to

$$\begin{aligned} \frac{dT_f}{dx} &= -\beta_1 \frac{US_1}{WC_{pf}} (T_g - T_f), \\ \frac{dT_g}{dx} &= -\beta_1 \frac{US_1}{WC_{pg}} (T_g - T_f) + \frac{(-\Delta H)S_2}{WC_{pg}} f_a \\ &\quad \times \left( \beta_2 K_1 \frac{1.5 p_{\text{N}_2} p_{\text{H}_2}}{p_{\text{NH}_3}} - \beta_3 K_2 \frac{p_{\text{NH}_3}}{1.5 p_{\text{H}_2}} \right), \\ \frac{dN_{\text{N}_2}}{dx} &= -f_a \left( \beta_4 K_1 \frac{1.5 p_{\text{N}_2} p_{\text{H}_2}}{p_{\text{NH}_3}} - \beta_5 K_2 \frac{p_{\text{NH}_3}}{1.5 p_{\text{H}_2}} \right). \end{aligned}$$

Thus, the variables for the optimization problem are  $\beta_i$  for  $i = 1, 2, 3, 4, 5$ . If the positivity or negativity of some terms in the expressions must be ensured, adequate variable bounds can be added for the respective  $\beta_i$  variable.

### 1.2.3 Data Fitting and Regression

Data fitting and regression analysis are activities commonly used by scientists, engineers and managers. We often employ a software to help us find a fitted curve on a set of data. If thought carefully, the software formulates and solves an optimization problem to arrive at the fitted curve. For a set of given data points (say  $(\mathbf{x}^{i,p}, y^{i,p})$  for  $i = 1, 2, \dots, K$ ) involving input variable set  $\mathbf{x}$  and output  $y$ , any curve  $y = f(\mathbf{x})$  is assigned an objective function as the sum of the squared difference between the  $y^{i,p}$  and  $f(\mathbf{x}^{i,p})$ :

$$\text{Minimize} \quad \sum_{i=1}^K (f(\mathbf{x}^{i,p}) - y^{i,p})^2. \quad (1.9)$$

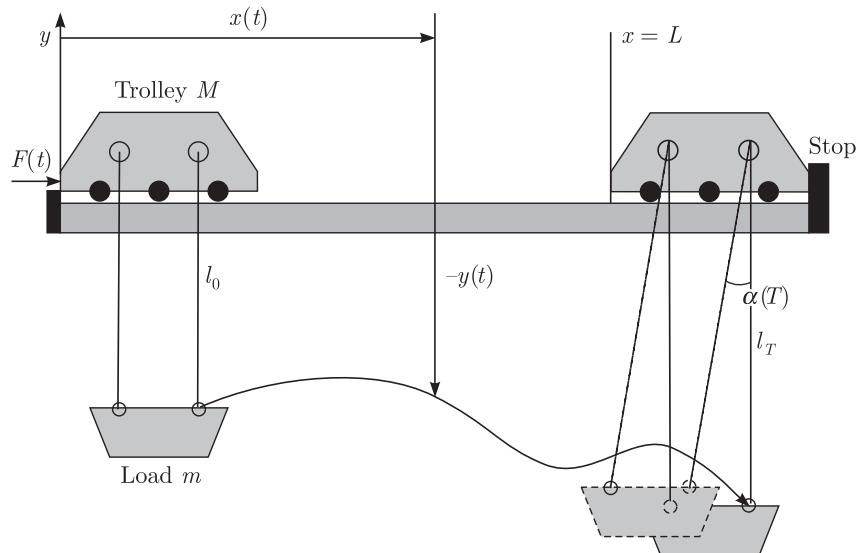
The decision variables are the coefficients of the fitted curve  $f(\mathbf{x})$ . For a single-input linear regression ( $y = ax + b$ ), there are two decision variables  $a$  and  $b$ . The minimization procedure will find  $a$  and  $b$  values that will make the overall error between the fitted line and the supplied data points minimum. The same idea can be extended to fit a curve of any sort. The regression task can be made more generic, by keeping the nature of the curve  $y = f(\mathbf{x})$  also as a part of the variable set. This domain then falls under the category of modelling of systems which is discussed in the previous subsection.

### 1.2.4 Control Systems

An optimal control problem involves decision parameters that change with time or distance or some other varying quantities. The optimization task is to find variations of decision parameters so as to minimize a performance metric or maximize quality of output quantities of the system. We consider an example here.

Overhead cranes are used in the industries, workshops, factories, docks, and other places to transport heavy components from one place to another. In order to increase the productivity, such control operations must be optimized to find what speed the overhead trolley must be moved so that the supplied energy to the crane and the overall operation time are minimum.

Figure 1.7 shows a schematic model of the crane (Deb and Gupta, 2004). In this simplified model, the cable connecting the trolley and the hanging load is considered of fixed length; however, in practice such cables are varied in length while moving in order to lower or raise the load (Dissanayake and Fang, 2001). The system has two degrees-of-freedom: (i)  $x$  denoting the linear motion of the trolley along the  $x$ -direction and (ii)  $\alpha$  denoting the angular motion of the hanging mass.



**Figure 1.7** A schematic of the overhead crane consisting of a trolley and a swaying load.

In the control system optimization problem, there are two time-varying control parameters: (i) a time-varying force  $F(t)$  applied to the trolley in the direction of the destination (along positive  $x$ -direction) and (ii) a time-dependent scheme of lowering the load from a high to a low altitude (with a difference of  $l_T$ ). Contrary to design optimization problems, here, the variable vector for the optimization problem is time-varying. Thus, instead of a single

value of force  $F$ , we need to identify a profile of  $F$  as a function of time  $t$  that would optimize the system.

By considering the force balances in  $x$  and  $y$  directions of all forces acted on the trolley, we obtain the following two equations:

$$M\ddot{x} = F + 2T \sin(\alpha) - \mu N,$$

$$N = Mg + 2T \cos(\alpha),$$

where  $T$  is twice the tension in each cable,  $M$  is the mass (in kg) of the trolley, and  $\ddot{x}$  is the acceleration of the trolley in the  $x$ -direction. Performing a similar task for the hanging load (of mass  $m$  kg), we have the following two equations:

$$-2T \sin(\alpha) - c\dot{x}_1 = m\ddot{x}_1,$$

$$2T \cos(\alpha) - c\dot{y}_1 - mg = m\ddot{y}_1,$$

where  $c$  is the coefficient of damping arising due to several factors on to the cable and the hanging mass. The variables  $x_1$  and  $y_1$  are the displacement of the hanging load in the  $x$  and  $y$  directions, respectively. In addition, the following relationships between trolley and the hanging load motions can be written with variables along  $x$  and  $y$  directions:

$$\begin{aligned} x_1 &= x + l_o \sin(\alpha), & y_1 &= -l_o \cos(\alpha), \\ \dot{x}_1 &= \dot{x} + l_o \dot{\alpha} \cos(\alpha), & \dot{y}_1 &= l_o \dot{\alpha} \sin(\alpha), \\ \ddot{x}_1 &= \ddot{x} + l_o \ddot{\alpha} \cos(\alpha) - l_o \dot{\alpha}^2 \sin(\alpha). & \ddot{y}_1 &= l_o \ddot{\alpha} \sin(\alpha) + l_o \dot{\alpha}^2 \cos(\alpha). \end{aligned}$$

Here,  $l_o$  is the length of the cable,  $\dot{\alpha}$  and  $\ddot{\alpha}$  are the angular velocity and acceleration of the cable. By eliminating  $T$ ,  $x_1$  and  $y_1$  from the above expressions, we get the following two equations of motion of the trolley and the hanging mass:

$$\begin{aligned} \ddot{x} &= [F - c\dot{x} \sin^2(\alpha) + ml_o \sin(\alpha) \dot{\alpha}^2 + mg \sin(\alpha) \cos(\alpha) \\ &\quad - f(ml_o \cos(\alpha) \dot{\alpha}^2 - c\dot{x} \sin(\alpha) \cos(\alpha) - mg \sin^2(\alpha))] \\ &\quad / (M + m \sin^2(\alpha) - fm \sin(\alpha) \cos(\alpha)), \end{aligned} \tag{1.10}$$

$$\ddot{\alpha} = -(\ddot{x} + r\dot{x} + g \tan(\alpha)) \frac{\cos(\alpha)}{l_o} - r\dot{\alpha}, \tag{1.11}$$

where  $r$  is the ratio of  $c$  to  $m$ . These two equations can be solved using a numerical integration technique and the variation of  $x$  and  $\alpha$  with time  $t$  can be found.

A little thought of the problem makes it clear that the two objectives (i) total energy supplied to the system and (ii) the total time for the block-load system to reach the desired position and stabilize are the two conflicting

objectives. The supplied energy will be minimum for the case of moving ever slowly towards the destination. But such a solution will require quite a long time to complete the task. On the other hand, reaching the destination with a large velocity and suddenly stopping at the destination would be a quick way to reach the destination; however, some time needs to be elapsed for the sway of the load to diminish. Although such a solution may not be the quickest overall time solution, there would exist a solution with a reasonable velocity which would minimize the overall time. Ideally, this is a two-objective problem, but for the sake of our discussion here, we may formulate a single-objective time minimization of the crane maneuvering problem:

$$\text{Minimize} \quad \text{Time} = T,$$

subject to

$$(x(T), y(T))^T = (L, l_T)^T,$$

$$\alpha(T) \leq \epsilon.$$

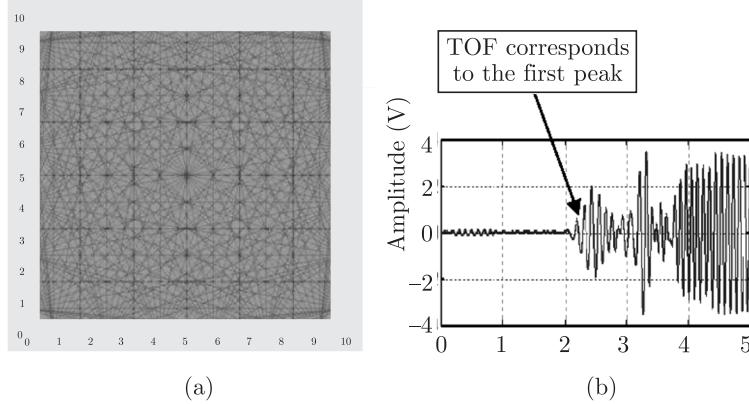
Values  $x(T)$  and  $\alpha(T)$  can be found by solving the differential equations given in Equations (1.10) and (1.11) from suitable initial conditions. The parameter  $\epsilon$  is a user-specified small sway angle for a stable system at time  $T$ . The decision variables of this problem are  $F(t)$  and  $\ell(t)$ .

### 1.2.5 Inverse Problems

Many problems in practice are solved as an inverse problem, as the forward problem in such cases is difficult to formulate and solve. For example, in a number of practical applications it is not only desirable but necessary to estimate the internal structure of a given object without destroying the object's internal structure. For example, in medical science applications, one is interested in diagnosis of the health of internal organs in a non-invasive manner. Similarly, an engineer is interested in estimating the presence of internal defects/inclusions inside a structure by what is called non-destructive evaluation (NDE) methods. Reconstruction of the cross-sectional view of an object, without destroying the object's structure, forms the basis for computed tomography. Tomographic reconstruction is an inverse problem wherein the task is to obtain the parameters of a model from observed or measured data.

Tomography makes use of various physical phenomena for collection of data which serves as input to the reconstruction algorithm chosen. The selection of a specific physical phenomena depends on the materials in the specimen being evaluated. Ultrasonic tomography (UT) has been in use for a long period now and offers the most convenient method for flaw detection in a material (Rose, 2002). To simulate the time of flight (TOF) of ultrasound rays, the specimen under consideration is represented as a grid of certain integer values corresponding to different materials. A number of ultrasound sources and detectors are positioned around the specimen in what is termed

as modified cross-hole geometry (Khare et al., 2007). For any configuration, the sources are actuated in sequence one at a time, and from each of these ultrasound rays travel to each of the detectors, giving us actual TOF data. For example, considering 6 sources, 6 detectors and 6 configurations, we get  $R = 216$  readings, which is essentially the input to our reconstruction algorithm. The ray coverage for this case is shown in Figure 1.8(a). The TOF is taken to be the arrival time corresponding to the first peak of the signal sensed by the detector as shown in Figure 1.8(b). If experimental data is available for these set of sources and detectors, the same can be used. Otherwise, the TOF can be calculated by using the following procedure.



**Figure 1.8** (a) Ray coverage, (b) Ultrasound signal indicating TOF.

The simulation of TOF assumes that the ultrasound rays follow straight paths from source to the detector. This assumption is reasonable when the impedance mismatch within the specimen is small.

A solution to this problem is a configuration in which some cells are occupied by foreign materials or voids (Kodali et al., 2008). The task of the optimization is then to identify the cells with foreign materials or voids that minimize the error between actual TOF and simulated TOF. The simulated TOF for a ray  $r$  originating from  $j$ -th source and terminating at the  $k$ -th detector is estimated using Equation (1.12):

$$\text{TOF}^{\text{simulated}}(r) = \sum_{m=1}^M \frac{l_m^{(r)}(j, k)}{v_m}, \quad (1.12)$$

where

$M$  = number of cells intercepted by the ray,

$l_m^{(r)}(j, k)$  = length of ray intercepted by  $m$ -th cell along the ray path,

$v_m$  = velocity of propagation of ultrasound through the  $m$ -th cell.

The process of reconstruction is an optimization task which consists of two major steps, the first is acquisition of TOF data and the second is using the

acquired data to reconstruct the specimen under consideration. The following optimization problem can be formulated:

$$\text{Minimize} \quad \text{Error} = \sum_{r=1}^R \left( \text{TOF}^{\text{simulated}}(r) - \text{TOF}^{\text{actual}}(r) \right)^2, \quad (1.13)$$

where  $R$  is the number of rays that are considered. The optimization procedure will then enable finding the size and location of voids or foreign material embedded in a component.

### 1.2.6 Scheduling and Routing

Scheduling and routing problems are different kinds of optimization tasks than what we have discussed so far. They are popularly faced in practice. For example, cases such as finding an optimal schedule for a classroom timetabling, or an examination scheduling, or airline time-tabling, or nurse timetabling in hospitals are some such problems that are commonly faced in practice. Job-shop scheduling in which the different machining tasks (or jobs) need to be performed on a set of machines and the optimization task is to find a schedule of tasks so that overall time of completion of the machining process is minimum.

A travelling salesperson problem is described in Figure 1.9. A salesperson has to travel to various cities to sell a product. The salesperson will travel to each city only once and must visit all given cities. In such a problem, the task is to travel to all given cities in such a way that will minimize the overall distance covered by the salesperson. Instead of representing variables as  $x_i$ , in scheduling problems, the *edge* between two quantities is important. For example, if the salesperson has to travel between three cities A, B, and C, alternative solutions are as follows: ABC, ACB, BAC, BCA, CAB and CBA. The solution ABC means that the salesperson will start from city A and then move to city B and then to city C. In these solutions, the absolute location of city A, B, or C is unimportant, rather the edge between A and B or B and A is an important matter. This book does not address the scheduling optimization problems; however, more information can be found in related texts (Brucker, 2010; Pinedo, 2008).

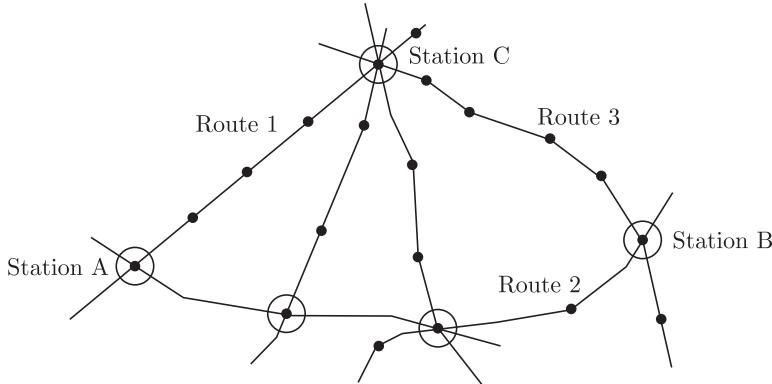
### Optimal design of a transit schedule

Figure 1.10 shows a typical transit system network. The solid lines represent different routes, the points on the lines represent the stops and the circled intersections of the routes represent the transfer stations. The problem is to determine schedules for the routes such that the transit system provides the best level of service (LOS) to its passengers, within the resources available. One of the good measures of the LOS is the amount of time passengers wait during their journey—the lesser the waiting time, the better is the LOS (Chakroborty et al., 1995). On any transit network, either the passengers wait to board the vehicle at the station of origin or they wait at a transfer



**Figure 1.9** A typical schedule for a travelling salesperson problem with some Indian cities.

station at which they transfer from one vehicle to another. For example, a passenger wishing to travel from station A to station B (in the network shown in Figure 1.10) will have to wait at station A to board a vehicle on Route 1. Further, the passenger will have to wait at transfer station C to board a vehicle on Route 3 (which will take him/her to the destination). We will refer to the wait at station A as the initial wait time (IWT) and the wait at station C as the transfer time (TT). A good schedule is one which minimizes the sum of IWT and TT for all passengers. Thus, the optimization problem involves finding a schedule of vehicles on all routes (arrival and departure times) such that the total waiting time for the passengers is minimum.



**Figure 1.10** A typical transit system network.

The design variables in this problem are the arrival time  $a_i^k$  and departure time  $d_i^k$  for the  $k$ -th vehicle at the  $i$ -th route. Thus, if in a problem, there are a total of  $M$  routes and each route has  $K$  vehicles, the total number of design variables is  $2MK$ . In addition, there are a few more artificial variables which we shall discuss later.

The constraints in this problem appear from different service-related limitations. Some of these constraints are formulated in the following:

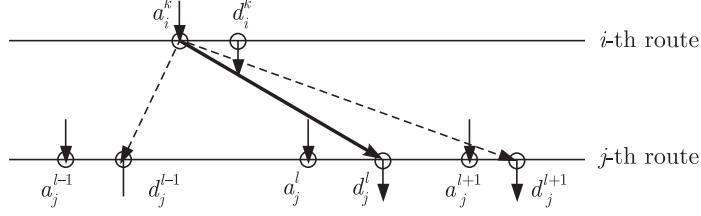
*Minimum stopping time:* A vehicle cannot start as soon as it stops; it has to wait at the stop for a certain period of time, or

$$(d_i^k - a_i^k) \geq s_{\min} \quad \text{for all } i \text{ and } k.$$

*Maximum stopping time:* A vehicle cannot stop for more than a certain period of time even if it means increasing the total transfer time on the network, or

$$(d_i^k - a_i^k) \leq s_{\max} \quad \text{for all } i \text{ and } k.$$

*Maximum allowable transfer time:* No passenger on the transit network should have to wait more than a certain period of time  $T$  at any transfer station. This can be enforced by checking all possible differences between departure and arrival times and limiting those values to  $T$ . Mathematically, this is difficult to achieve. We simplify the formulation of this constraint by introducing a new set of variables  $\delta_{i,j}^{k,l}$  between the  $k$ -th vehicle of the  $i$ -th route and the  $l$ -th vehicle of the  $j$ -th route. These variables can take either a zero or a one. A value of zero means that the transfer of passengers between those two vehicles is not feasible. A value of one means otherwise. Consider the arrival and departure times of vehicles in two different routes at a particular station, as shown in Figure 1.11. The parameters  $a_i^k$  and  $d_i^k$  are the arrival and departure times of the  $k$ -th vehicle in the  $i$ -th route. A passenger from the  $k$ -th vehicle in the  $i$ -th route can only transfer to a vehicle in the  $j$ -th route, which is arriving at the station after  $a_i^k$ . According to the figure, the transfer of a passenger from the  $k$ -th vehicle



**Figure 1.11** Transfers from the  $k$ -th vehicle in the  $i$ -th route to three consecutive vehicles in the  $j$ -th route.

in the  $i$ -th route is not possible to the  $(l-1)$ -th vehicle in the  $j$ -th route, because the departure time of the latter vehicle ( $d_j^{l-1}$ ) is earlier than that of  $a_i^k$ . Thus, the parameter  $\delta_{i,j}^{k,l-1}$  takes a value zero, whereas the parameter  $\delta_{i,j}^{k,l}$  takes a value one. In order to simplify the model, we assume that transfers to vehicles departing after the  $l$ -th vehicle in the  $j$ -th route are also not possible. All parameters  $\delta_{i,j}^{k,q}$  for  $q = (l+1), (l+2), \dots$  are also zero. Thus, between any two vehicles, the following condition must be satisfied:

$$(d_j^l - a_i^k)\delta_{i,j}^{k,l} \leq T \quad \text{for all } i, j, k \text{ and } l.$$

It is clear that the left side expression of the above condition is zero for those transfers that are not feasible. Since transfers only to the next available vehicle are assumed, only one  $\delta_{i,j}^{k,l}$  (for  $l = 1, 2, \dots$ ) is one and the rest all are zeros for fixed values of  $i$ ,  $j$ , and  $k$ . Mathematically,

$$\sum_l \delta_{i,j}^{k,l} = 1 \quad \text{for all } i, j \text{ and } k.$$

The introduction of the artificial variables  $\delta_{i,j}^{k,l}$  makes the formulation easier, but causes a difficulty. Many optimization algorithms cannot handle discrete design variables efficiently. Since the artificial design variables  $\delta_{i,j}^{k,l}$  can only take a value zero or one, another set of constraints is added to enforce the binary values:

$$(d_j^l - a_i^k) + M(1 - \delta_{i,j}^{k,l}) \geq 0 \quad \text{for all } i, j, k \text{ and } l,$$

where  $M$  is a large positive number. The above constraint ensures that the variable  $\delta_{i,j}^{k,l}$  always takes a value one whenever a transfer is possible and the value zero whenever a transfer is not possible. This constraint is derived purely from the knowledge of the available optimization algorithms. There may be other ways to formulate the concept of feasible transfers, but inclusion of such artificial design variables often makes the understanding of the problem easier.

*Maximum headway:* The headway between two consecutive vehicles should be less than or equal to the policy headway,  $h_i$ , or

$$(a_i^{k+1} - a_i^k) \leq h_i \quad \text{for all } i \text{ and } k.$$

The objective function consists of two terms: the first term represents the total transfer time (TT) over all the passengers and the second term represents the initial waiting time (IWT) for all the passengers. The objective is to minimize the following function:

$$\begin{aligned} & \sum_i \sum_j \sum_k \sum_l \delta_{i,j}^{k,l} (d_j^l - a_i^k) w_{i,j}^k \\ & + \sum_i \sum_l \int_0^{a_i^k - a_i^{k-1}} v_{i,k}(t) [(a_i^k - a_i^{k-1}) - t] dt. \end{aligned}$$

The parameter  $w_{i,j}^k$  is the number of passengers transferring from the  $k$ -th vehicle of the  $i$ -th route to the  $j$ -th route. The first term is obtained by summing the individual transfer time ( $d_j^l - a_i^k$ ) over all passengers for all the vehicles for every pair of routes. The parameter  $v_{i,k}(t)$  is the number of passengers arriving at the stop for the  $k$ -th vehicle in the  $i$ -th route at a given time  $t$ . Since the arrival time for passengers can be anywhere between  $t = 0$  and  $t = (a_i^k - a_i^{k-1})$  (the headway), the initial waiting time also differs from one passenger to another. For example, a passenger arriving at the stop just after the previous vehicle has left has to wait for the full headway time ( $a_i^k - a_i^{k-1}$ ) before the next vehicle arrives. On the other hand, a passenger arriving at the stop later has to wait for a shorter time. The calculation of the second term assumes that passengers arrive at the stop during the time interval  $a_i^{k-1}$  to  $a_i^k$  according to the known time-varying function  $v_{i,k}(t)$ , where  $t$  is measured from  $a_i^{k-1}$ . Then the quantity

$$\int_0^{a_i^k - a_i^{k-1}} v_{i,k}(t) [(a_i^k - a_i^{k-1}) - t] dt$$

gives the sum of the initial waiting times for all passengers who board the  $k$ -th vehicle of the  $i$ -th route. We then sum it over all the routes and vehicles to estimate the network total of the IWT. Thus, the complete NLP problem can be written as follows:

$$\begin{aligned} \text{Minimize} \quad & \sum_i \sum_j \sum_k \sum_l \delta_{i,j}^{k,l} (d_j^l - a_i^k) w_{i,j}^k \\ & + \sum_i \sum_l \int_0^{a_i^k - a_i^{k-1}} v_{i,k}(t) [(a_i^k - a_i^{k-1}) - t] dt \end{aligned}$$

subject to

$$s_{\max} - (d_i^k - a_i^k) \geq 0 \quad \text{for all } i \text{ and } k,$$

$$(d_i^k - a_i^k) - s_{\min} \geq 0 \quad \text{for all } i \text{ and } k,$$

$$T - (d_j^l - a_i^k) \delta_{i,j}^{k,l} \geq 0 \quad \text{for all } i, j, k \text{ and } l,$$

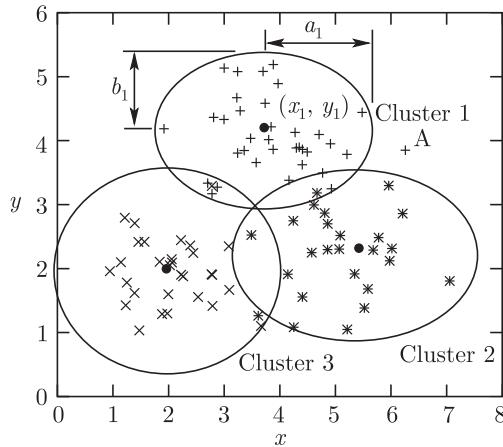
$$\begin{aligned}
(d_j^l - a_i^k) + M(1 - \delta_{i,j}^{k,l}) &\geq 0 \quad \text{for all } i, j, k \text{ and } l, \\
h_i - (a_i^{k+1} - a_i^k) &\geq 0 \quad \text{for all } i \text{ and } k, \\
\sum_l \delta_{i,j}^{k,l} &= 1 \quad \text{for all } i, j \text{ and } k.
\end{aligned}$$

In the above NLP problem, the variables  $\delta_{i,j}^{k,l}$  are binary variables taking only a value zero or a one and other variables  $a_i^k$  and  $d_i^k$  are real-valued. Thus, a mixed integer programming technique described in Chapter 5 or genetic algorithms described in Chapter 6 can be used to solve the above NLP problem (Chakroborty et al., 1995).

### 1.2.7 Data Mining

Data mining activities have become an inseparable part of any scientific work, mainly due to the availability and ease of generation of data. When the data generation task is expensive or tedious, data is usually generated in parallel among various research groups and they are shared by scientists. In such an activity, the first task is often to cluster the data into several categories to establish which data from what source are related to each other. Optimization is usually a tool employed for such a clustering activity. Here, we discuss such a procedure.

Let us consider a two-dimensional data shown in Figure 1.12. When the data is achieved, they are all jumbled together. One way to formulate an optimization problem is to first decide on a number of clusters that the data may be divided to. Then, for each cluster, a few parameters are used variables for the optimization task. The parameters describe an affinity function for a cluster. Say, for example, we define a simple affinity function for the  $i$ -th cluster, as follows:



**Figure 1.12** Clustering procedure is illustrated.

$$c_i(x, y) = \frac{(x - x_i)^2}{a_i^2} + \frac{(y - y_i)^2}{b_i^2}.$$

This is an equation of an ellipse with centre at  $(x_i, y_i)$  and major and minor axis values as  $a_i$  and  $b_i$ , respectively. Next, each data point  $(x, y)$  is tested for each cluster affinity function  $C_i$ . The point is associated with that cluster for which the affinity function value is minimum. This procedure will associate every data point with a particular cluster. Point A (in the figure) may get associated with Cluster 2.

Next, an intra-cluster distance ( $D_i^{\text{intra}}$ ) between all associated points of each cluster is computed as follows:

$$D_i^{\text{intra}} = \frac{1}{|C_i|} \left( \sum_{(x,y) \in C_i} \sqrt{(x - x_i)^2 + (y - y_i)^2} \right). \quad (1.14)$$

Further, an inter-cluster distance ( $D_i^{\text{inter}}$ ) is computed as follows:

$$D_i^{\text{inter}} = \frac{1}{\binom{K}{2}} \left( \sum_{i=1}^K \sum_{j=1, j \neq i}^K \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \right). \quad (1.15)$$

Then, the following optimization problem is solved:

$$\text{Minimize } \frac{D_i^{\text{intra}}}{D_i^{\text{inter}}},$$

subject to

$$(x_i, y_i) \in R,$$

where  $R$  is the specified range in which the centres of ellipses are supposed to lie. The minimization will ensure that intra-cluster points are as close to the centre as possible and all cluster centres are as far away from each other as possible, thereby achieving the goal of clustering.

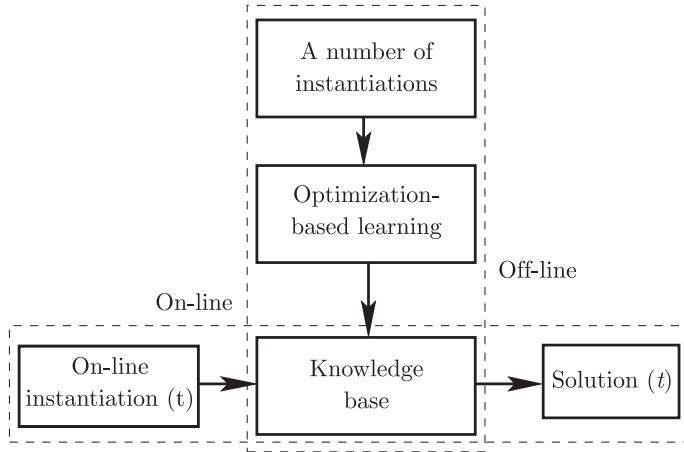
Prediction is another data mining activity which is used on a daily basis by weather forecasters, share market predictors, etc. The task here is to model the past few days (or weeks or months) of data and then extrapolate the model to predict how the future is going to be for the next hours, days or weeks. Since the modelling activity can be achieved through an optimization problem-solving (as discussed before), the prediction activity can be achieved using an optimization procedure.

### 1.2.8 Intelligent System Design

Many systems in practice are required to be designed in a manner that may show certain intelligence while performing a task. Since optimization

algorithms work at finding the optimal solution in the search space, which is not any arbitrary solution, they can be ideal candidates for arriving at an intelligent solution. For example, consider the robot path planning problem in which the task is to design the ‘brain’ of the robot which will help it to avoid obstacles that may be moving and still reach a destination with minimum time or using minimum energy or optimizing some other criterion.

The use of an optimization method to solve such problems causes an inherent problem. The optimal path depends on how the obstacles will be moving in a real scenario and unfortunately this information may not be known *a priori*. One way to address such a task is to first develop an optimal ‘rule base’ by performing an off-line optimization on several training cases. Once the optimal rule-base is developed, the robot can then utilize it to navigate in a real scenario. If sufficient training cases are considered, a fairly robust rule-base can be developed. Figure 1.13 depicts the above-discussed off-line optimization-based approach (Pratihar et al., 1999). On a set of instantiations, an optimization algorithm is applied to find a knowledge base using rules or by other means. The optimization task would find a set of rules or



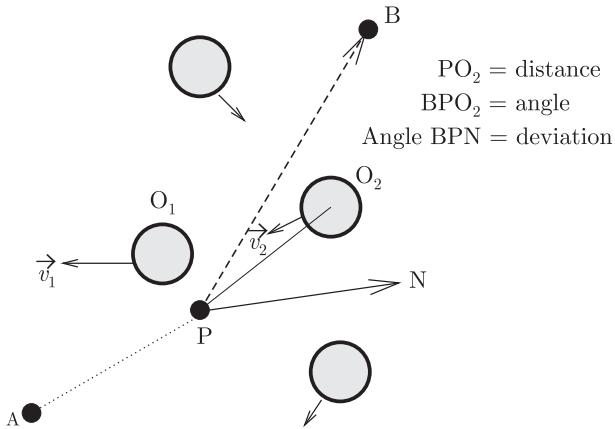
**Figure 1.13** Off-line optimization procedure is illustrated.

classifiers which will determine the nature of the outcome based on the variable values at any time instant. In the following, we describe the procedure in the context of an on-line robot navigation problem. There are essentially two parts of the problem:

- (i) Learn to find *any* obstacle-free path from point A to B, and
- (ii) Learn to choose that obstacle-free path which takes the robot in a minimum possible time.

This process of avoiding an object can be implemented using a rule of the following sort:

If an object is **very near** and is **approaching**, then turn **right** to the original path.



**Figure 1.14** A schematic showing condition and action variables for the robot navigation problem.

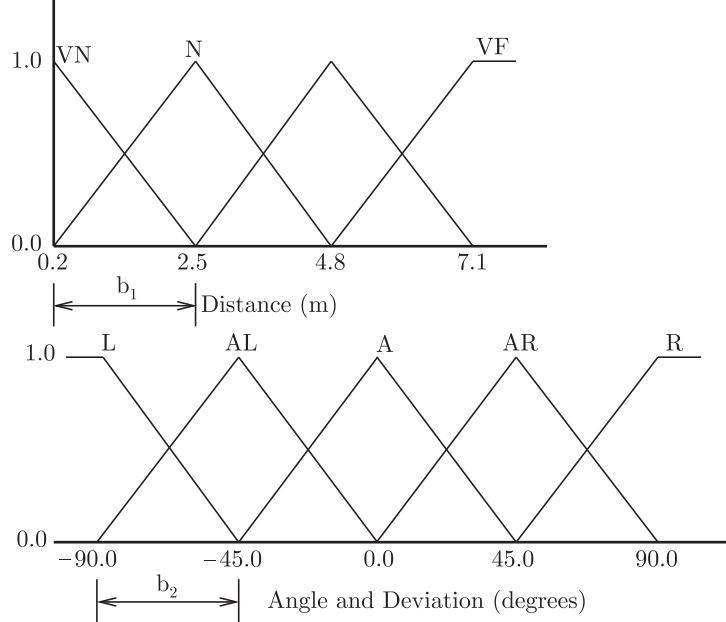
The angle and distance of a robot from a critical obstacle and its decision on deviation, if any, are illustrated in Figure 1.14. Because of the imprecise definition of the deviation in this problem, it seems natural to use a fuzzy logic technique (Mendel, 2000) here. Table 1.2 shows a set of possible fuzzy rules. The (1,1) position of the table suggests that if an obstacle is very near (VN) to the robot and it is left (L) of the robot, the decision is to move ahead (A).

**Table 1.2** A possible rule-base

		Angle				
		L	AL	A	AR	R
Distance	VN	A	AR	AL	AL	A
	N	A	A	AL	A	A
	F	A	A	AR	A	A
	VF	A	A	A	A	A

This way, the rule-base shows every possible decision that the robot should take in navigating in a real scenario. Finding what decision to take in every combination of angle and distance is the task of an off-line optimization algorithm. To extend the scope, the presence or absence of a rule for every combination of angle and distance may be considered as another kind of decision variable. The fuzzy rules involve membership functions that define the attributes such as the extent of distance values for VN, N, etc. and also A, L, etc. Triangular membership functions are commonly used for this purpose.

This involves another set of decision variables as the width of the triangles (such as parameters  $b_1$  and  $b_2$  shown in Figure 1.15). Thus, when a rule-



**Figure 1.15** A possible set of membership functions.

base (Table 1.2) is specified completely, it can be used to simulate a robot for different training scenarios. The overall time required from start to finish can then be recorded from the simulation process and whether the robot collides with any obstacle or not can be determined. The resulting off-line optimization problem then becomes as follows:

$$\text{Minimize Time (rule-base),}$$

$$\text{subject to} \quad \text{Path (rule-base) is obstacle-free.}$$

Note that both Time and Path are functions of the rule-base and the elements of the rule-base are decision variables of the optimization task.

### 1.3 Classification of Optimization Algorithms

The above optimization problems reveal the fact that the formulation of engineering design problems could differ from problem to problem. Certain problems involve linear terms for constraints and objective function but certain other problems involve nonlinear terms for them. In some problems, the terms are not explicit functions of the design variables. Unfortunately, there does not exist a single optimization algorithm which will work in all optimization problems equally efficiently. Some algorithms perform better on

one problem, but may perform poorly on other problems. That is why the optimization literature contains a large number of algorithms, each suitable to solve a particular type of problem. The choice of a suitable algorithm for an optimization problem is, to a large extent, dependent on the user's experience in solving similar problems. This book provides a number of optimization algorithms used in engineering design activities.

Since the optimization algorithms involve repetitive application of certain procedures, they need to be used with the help of a computer. That is why the algorithms are presented in a step-by-step format so that they can be easily coded. To demonstrate the ease of conversion of the given algorithms into computer codes, most chapters contain a representative working computer code. Further, in order to give a clear understanding of the working of the algorithms, they are hand-simulated on numerical exercise problems. Simulations are performed for two to three iterations following the steps outlined in the algorithm sequentially. Thus, for example, when the algorithm suggests to move from Step 5 to Step 2 in order to carry out a conditional statement, the exercise problem demonstrates this by performing Step 2 after Step 5. For the sake of clarity, the optimization algorithms are classified into a number of groups, which are now briefly discussed.

*Single-variable optimization algorithms.* Because of their simplicity, single-variable optimization techniques are discussed first. These algorithms provide a good understanding of the properties of the minimum and maximum points in a function and how optimization algorithms work iteratively to find the optimum point in a problem. The algorithms are classified into two categories—direct methods and gradient-based methods. Direct methods do not use any derivative information of the objective function; only objective function values are used to guide the search process. However, gradient-based methods use derivative information (first and/or second-order) to guide the search process. Although engineering optimization problems usually contain more than one design variable, single-variable optimization algorithms are mainly used as unidirectional search methods in multivariable optimization algorithms.

*Multivariable optimization algorithms.* A number of algorithms for unconstrained, multivariable optimization problems are discussed next. These algorithms demonstrate how the search for the optimum point progresses in multiple dimensions. Depending on whether the gradient information is used or not used, these algorithms are also classified into direct and gradient-based techniques.

*Constrained optimization algorithms.* Constrained optimization algorithms are described next. These algorithms use the single-variable and multivariable optimization algorithms repeatedly and simultaneously maintain the search effort inside the feasible search region. Since these algorithms are mostly used in engineering optimization problems, the discussion of these algorithms covers most of the material of this book.

*Specialized optimization algorithms.* There exist a number of structured algorithms, which are ideal for only a certain class of optimization problems. Two of these algorithms—integer programming and geometric programming—are often used in engineering design problems and are discussed. Integer programming methods can solve optimization problems with integer design variables. Geometric programming methods solve optimization problems with objective functions and constraints written in a special form.

There exist quite a few variations of each of the above algorithms. These algorithms are being used in engineering design problems since sixties. Because of their existence and use for quite some years, we call these algorithms as traditional optimization algorithms.

*Nontraditional optimization algorithms.* There exist a number of other search and optimization algorithms which are comparatively new and are becoming popular in engineering design optimization problems in the recent past. Two such algorithms—genetic algorithms and simulated annealing—are discussed in this book.

We have put together about 36 different optimization algorithms. Over the years, researchers and practitioners have modified these algorithms to suit their problems and to increase the efficiency of the algorithms. However, there exist a few other optimization algorithms—stochastic programming methods and dynamic programming method—which are very different than the above algorithms. Because of the space limitation and occasional use of these algorithms in engineering design problems, we have not included them in this book. A detailed discussion of these algorithms can be found elsewhere (Rao, 1984).

Many engineering optimization problems contain multiple optimum solutions, among which one or more may be the absolute minimum or maximum solutions. These absolute optimum solutions are known as global optimal solutions and other optimum solutions are known as local optimum solutions. Ideally, we are interested in the global optimal solutions because they correspond to the absolute optimum objective function value. An algorithm that has a perspective of finding an optimal solution close to the initial starting solution is known as a *local search* algorithm. Most traditional optimization algorithms described in this book have this local search property. Starting with a single initial point, the algorithms try to find an optimal solution for which either the derivative vector is close to a zero vector or the KKT necessary conditions are satisfied. These algorithms do not guarantee finding the global optimal solution. However, nontraditional algorithms, such as genetic algorithms and simulated annealing algorithm, are found to have a better global perspective than the traditional methods. The global optimality issues are discussed in Chapter 6.

Moreover, when an optimal design problem contains multiple global solutions, designers are interested in finding not only just one global optimum solution, but as many as possible for various reasons. The optimization

algorithms that attempt to find multiple optimal solutions are called *multimodal optimization* algorithms (Deb and Goldberg, 1989). There are several practical reasons for finding multiple optimal solutions. Firstly, a design suitable in one situation may not be valid in another situation. Secondly, it is also not possible to include all aspects of the design in the optimization problem formulation. Thus, there always remains some uncertainty about the obtained optimal solution. Thirdly, designers may not be interested in finding the absolute global solution, instead may be interested in a solution which corresponds to a marginally inferior objective function value but is more amenable to fabrication. Thus, it is always prudent to know about other equally good solutions for later use. However, if the traditional methods are used to find multiple optimal solutions, they need to be applied a number of times, each time starting from a different initial solution and hoping to achieve a different optimal solution each time. Genetic algorithms described in Chapter 6 allow an easier way to find multiple optimal solutions simultaneously in a single simulation.

Another class of optimization problems deals with simultaneous optimization of multiple objective functions. In formulating an optimal design problem, designers are often faced with a number of objective functions. For example, the truss structure problem described earlier should really be reformulated as the minimization of both the weight of the truss and the deflection at the point  $C$ . Multiobjective optimization problems give rise to a set of optimal solutions known as *Pareto-optimal* solutions (Chankong and Haimes, 1983), all of which are equally important as far as all objectives are concerned. Thus, the aim in these problems is to find as many Pareto-optimal solutions as possible. Because of the complexity involved in the multiobjective optimization algorithms, designers usually choose to consider only one objective and formulate other objectives as constraints. Genetic algorithms described in Chapter 6 demonstrate one way to handle multiple objectives and help find multiple Pareto-optimal solutions simultaneously.

Many engineering design problems have a multi-level structure. For example, in a bi-level optimization problem, there are two levels of optimization problems. An optimal solution to the lower-level optimization problem becomes a feasible solution to the upper-level optimization problem. In such a *nested optimization* task, in which the evaluation of a solution on the upper-level optimization problem requires solving another optimization problem (lower-level problem), unless the optimal solution of the lower-level problem is found, the evaluation of the upper-level solution is meaningless. A broom-balancing problem is a good example of a bi-level optimization problem. The strategy to control the motion of the platform for minimum energy requirement is the upper-level optimization problem and once a strategy is given, finding whether the broom will get balanced in a minimum time is the lower-level optimization problem. Thus, unless the broom is balanced and it does it in the minimum possible time, the strategy cannot be evaluated to find the energy it demands in the upper-level. Such problems are common in engineering and efficient optimization algorithms are needed

to solve such problems (Deb and Sinha, 2010). Tri-level and higher-level optimization problems are also possible.

Most optimization problems discussed in this chapter are deterministic in nature, meaning that the objective function, constraint function or parameters are fixed from the beginning till the end of the optimization run. These problems are called *deterministic optimization* problems. However, as discussed somewhat in Section 1.2.8, some problems may have dynamically changing objective function or constraint function or parameters. While the optimization run is in process, one or a few of the functions and parameters may change. This happens usually if a system depends on the environment (ambient temperature or humidity) and the process continues over most of the day. The ambient condition becomes different from morning to noon. Thus, the solution that was optimal during morning may not remain optimal in the noon time. Such problems are called *stochastic optimization* problems, or sometimes *dynamic optimization* problems as well. Although an off-line optimization, as discussed in Section 1.2.8, is one approach, recent dynamic optimization methodologies (Deb et al., 2007) are other viable and pragmatic approaches.

At the end of the optimization process, one obvious question may arise: Is the obtained solution a true optimum solution? Unfortunately, there is no easy answer to this question for an arbitrary optimization problem. In problems where the objective functions and constraints can be written in simple, explicit mathematical forms, the Kuhn-Tucker conditions described in Chapter 4 may be used to check the optimality of the obtained solution. However, those conditions can only be used for a few classes of optimization problems. In a generic problem, this question is answered in a more practical way. In many engineering design problems, a good solution is usually known either from the previous studies or from experience. After formulating the optimal problem and applying the optimization algorithm if a better solution than the known solution is obtained, the new solution becomes the current best solution. The optimality of the obtained solution is usually confirmed by applying the optimization algorithms a number of times from different initial solutions.

Having discussed different types of optimization algorithms that exist to solve different kinds of optimization problems, we suggest that for practical applications there is always a need for *customizing* an optimization algorithm for solving a particular problem. The reasons are many. First, there may not exist a known efficient optimization algorithm for a particular problem at hand. Second, the size and complexity of solving a problem using an algorithm may be computationally too expensive for it to be pragmatic. Third, in many scenarios, the user may not be interested in finding the exact optimal solution, rather a reasonably good (or better than an existing solution) may be adequate. In such scenarios, knowledge (or problem information) about the current problem may be used to design a *customized optimization* algorithm that is appropriate for the current problem (Deb, Reddy and Singh, 2003). Such optimization methods are often known as *heuristic optimization*

methods. This requires that an algorithm is flexible to be updated with problem information. Mathematical optimization methods are difficult to update with heuristic information; however, nontraditional methods, such as genetic algorithms, simulated annealing, and others, are flexible in this regard and are getting increasingly applied in practice for this singular reason.

## 1.4 Summary

In order to use optimization algorithms in engineering design activities, the first task is to formulate the optimization problem. The formulation process begins with identifying the important design variables that can be changed in a design. The other design parameters are usually kept fixed. Thereafter, constraints associated with the design are formulated. The constraints may arise due to resource limitations such as deflection limitations, strength limitations, frequency limitations, and others. Constraints may also arise due to codal restrictions that govern the design. The next task is to formulate the objective function which the designer is interested in minimizing or maximizing. The final task of the formulation phase is to identify some bounding limits for the design variables.

The formulation of an optimization problem can be more difficult than solving the optimization problem. Unfortunately, every optimization problem requires different considerations for formulating objectives, constraints, and variable bounds. Thus, it is not possible to describe all considerations in a single book. However, many of these considerations require some knowledge about the problem, which is usually available with the experienced designers due to their involvement with similar other design problems.

The rest of the book assumes that the formulation of an optimization problem is available. Chapters 2 to 6 describe a number of different optimization algorithms—traditional and nontraditional—in step-by-step format. To demonstrate the working of each algorithm, hand-simulations on a numerical example problem are illustrated. Sample computer codes for a number of optimization algorithms are also appended to demonstrate the ease of conversion of other algorithms into similar computer codes.

## REFERENCES

- Brucker, P. (2010). *Scheduling algorithms*. Heidelberg, Germany: Springer.
- Chakroborty, P., Deb, K., and Subrahmanyam, P. (1995). Optimal scheduling of urban transit systems using genetic algorithms. *ASCE Journal of Transportation Engineering*, **121**(6), 544–553.
- Chankong, V. and Haimes, Y. Y. (1983). *Multiobjective Decision Making Theory and Methodology*. New York: North-Holland.

- Deb, K. and Datta, R. (2012). Hybrid evolutionary multi-objective optimization and analysis of machining operations. *Engineering Optimization*, **44**(6), 685–706.
- Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization, In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, (Washington DC, USA), 42–50.
- Deb, K. and Gupta, N. (2004). Optimal operating conditions for overhead crane maneuvering using multi-objective evolutionary algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2004* (Seattle, USA). 1042–1053. (Also Lecture Notes in Computer Science (LNCS) 3102).
- Deb, K., Khan, N. and Jindal, S. (2000). Optimal truss-structure design for multiple objectives. *Tenth National Seminar on Aerospace Structures*, 168–180.
- Deb, K., Rao, U. B., and Karthik, S. (2007). Dynamic multi-objective optimization and decision-making using modified NSGA-II: A case study on hydro-thermal power scheduling bi-objective optimization problems. *Proceedings of the Fourth International Conference on Evol. Multi-Criterion Optimization (EMO-2007)*, (LNCS, Springer) 803–817.
- Deb, K., Reddy, A. R., and Singh, G. (2003). Optimal scheduling of casting sequence using genetic algorithms. *Journal of Materials and Manufacturing Processes*, **18**(3), 409–432.
- Deb, K. and Saxena, V. (1997). Car suspension design for comfort using genetic algorithms. In Thomas Back (ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-97)*, 553–560.
- Deb, K. and Sinha, A. (2010). An efficient and accurate solution methodology for bilevel multi-objective programming problems using a hybrid evolutionary-local-search algorithm. *Evolutionary Computation Journal*, **18**(3), 403–449.
- Dissanayake, G. and Fang, G. (2001). Minimum-time trajectory for reducing load sway in quay-cranes. *Engineering Optimization*, **33**, 643–662.
- Haykin, S. S. (1999). *Neural Networks: A Comprehensive Foundation*. New Delhi: Prentice-Hall of India.
- Khare, S., Razdan, M., Munshi, P., Sekhar, B. V., and Balasubramaniam, K. (2007). Defect detection in carbon-fiber composites using Lamb-wave tomographic methods. *RNDE*, **18**, 101–119.
- Kodali, S. P., Bandaru, S., Deb, K., Munshi, P., and Kishore, N. N. (2008). Applicability of Genetic Algorithms to Reconstruction of Projected Data from Ultrasonic Tomography. *Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2008)*, (Atlanta, USA), 1705–1706.
- Mendel, J. M. (2000). *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*, New Delhi: Prentice-Hall of India.

- Murase, A., Roberts, H. L., and Converse, A. O. (1970). Optimal thermal design of an autothermal ammonia synthesis reactor. *Ind. Eng. Chem. Process Des. Develop.*, **9**, 503–513.
- Pinedo, M. (2008). *Scheduling: Theory, Algorithms, and Systems*. New York: Springer.
- Pratihar, D., Deb, K., and Ghosh, A. (1999). Fuzzy-genetic algorithms and time-optimal obstacle-free path generation for mobile robots. *Engineering Optimization*, **32**, 117–142.
- Quiza Sardiñas, R., Rivas Santana, M., and Alfonso Brindis, E. (2006). Genetic algorithm-based multi-objective optimization of cutting parameters in turning processes. *Engineering Applications of Artificial Intelligence*, **19**(2), 127–133.
- Rao, S. S. (1984). *Optimization Theory and Applications*. New Delhi: Wiley Eastern.
- Rose, J. L. (2002). A baseline and vision of ultrasonic guided wave inspection potential. *Journal of Pressure Vessel Technology*, **124**, 273–282.
- Rumelhart, D. E. and McClelland, J. L. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume I*. Cambridge, MA: MIT Press.
- Shigley, J. E. (1986). *Mechanical Engineering Design*. New York: McGraw-Hill.
- Timoshenko, S. (1986). *Strength of Materials, Part 1: Elementary Theory and Problems*. Delhi: CBS Publishers.
- Upreti, S. and Deb, K. (1997). Optimal design of an ammonia synthesis reactor using genetic algorithms. *Computers & Chemical Engineering*, **21**(1), 87–92.

## 2

# Single-variable Optimization Algorithms

---

We begin our discussion with optimization algorithms for single-variable and unconstrained functions. Single-variable optimization algorithms are taught in the first or the second-year level engineering courses and many students are therefore familiar with them. Since single-variable functions involve only one variable, the optimization procedures are simple and easier to understand. Moreover, these algorithms are repeatedly used as a subtask of many multi-variable optimization methods. Therefore, a clear understanding of these algorithms will help readers learn complex algorithms discussed in subsequent chapters.

The algorithms described in this chapter can be used to solve minimization problems of the following type:

$$\text{Minimize } f(x),$$

where  $f(x)$  is the objective function and  $x$  is a real variable. The purpose of an optimization algorithm is to find a solution  $x$ , for which the function  $f(x)$  is minimum. Two distinct types of algorithms are presented in this chapter: Direct search methods use only objective function values to locate the minimum point, and gradient-based methods use the first and/or the second-order derivatives of the objective function to locate the minimum point. Gradient-based optimization methods presented in this chapter do not really require the function  $f(x)$  to be differentiable or continuous. Gradients are computed numerically wherever required. Even though the optimization methods described here are for minimization problems, they can also be used to solve a maximization problem by adopting any of the following two procedures. In the first procedure, an equivalent dual problem ( $-f(x)$ ) is formulated and minimized. In this case, the algorithms described in this chapter can be directly used to solve a maximization problem. In the second procedure, the if-then-else clauses used in the algorithms have to be modified

to directly solve maximization problems. The first procedure is simpler to use and is therefore popular, whereas the second procedure is more elegant but requires a proper understanding of the working of the optimization algorithms. We first present the necessary and sufficient conditions for optimality. Then, we describe two bracketing algorithms, followed by direct and gradient-based search methods.

## 2.1 Optimality Criteria

Before we present conditions for a point to be an optimal point, we define three different types of optimal points.

(i) *Local optimal point*: A point or solution  $x^*$  is said to be a local optimal point, if there exists no point in the neighbourhood of  $x^*$  which is better than  $x^*$ . In the parlance of minimization problems, a point  $x^*$  is a locally minimal point if no point in the neighbourhood has a function value smaller than  $f(x^*)$ .

(ii) *Global optimal point*: A point or solution  $x^{**}$  is said to be a global optimal point, if there exists no point in the entire search space which is better than the point  $x^{**}$ . Similarly, a point  $x^{**}$  is a global minimal point if no point in the entire search space has a function value smaller than  $f(x^{**})$ .

(iii) *Inflection point*: A point  $x^*$  is said to be an inflection point if the function value increases locally as  $x^*$  increases and decreases locally as  $x^*$  reduces or if the function value decreases locally as  $x^*$  increases and increases locally as  $x^*$  decreases.

Certain characteristics of the underlying objective function can be exploited to check whether a point is either a local minimum or a global minimum, or an inflection point. Assuming that the first and the second-order derivatives of the objective function  $f(x)$  exist in the chosen search space, we may expand the function in Taylor's series at any point  $\bar{x}$  and satisfy the condition that any other point in the neighbourhood has a larger function value. It can then be shown that conditions for a point  $\bar{x}$  to be a minimum point is that  $f'(\bar{x}) = 0$  and  $f''(\bar{x}) > 0$ , where  $f'$  and  $f''$  represent the first and second derivatives of the function. The first condition alone suggests that the point is either a minimum, a maximum, or an inflection point, and both conditions together suggest that the point is a minimum. In general, the sufficient conditions of optimality are given as follows:

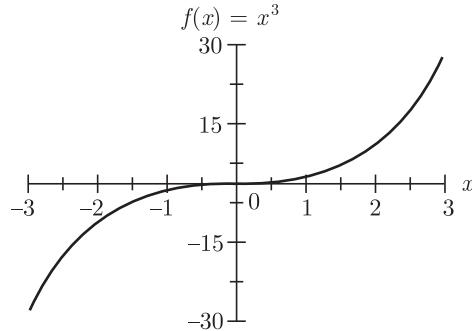
Suppose at point  $x^*$ , the first derivative is zero and the first nonzero higher order derivative is denoted by  $n$ ; then

- If  $n$  is odd,  $x^*$  is an inflection point.
- If  $n$  is even,  $x^*$  is a local optimum.
  - (i) If the derivative is positive,  $x^*$  is a local minimum.
  - (ii) If the derivative is negative,  $x^*$  is a local maximum.

We consider two simple mathematical functions to illustrate a minimum and an inflection point.

### EXERCISE 2.1.1

Consider the optimality of the point  $x = 0$  in the function  $f(x) = x^3$ . The function is shown in Figure 2.1. It is clear from the figure that the point



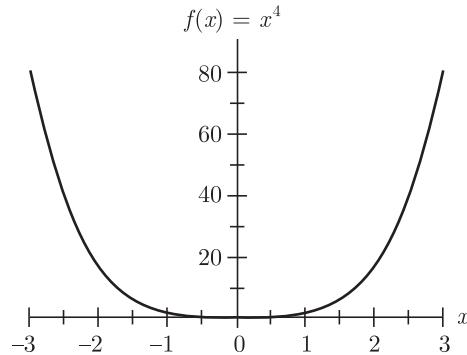
**Figure 2.1** The function  $f(x) = x^3$ .

$x = 0$  is an inflection point, since the function value increases for  $x \geq 0$  and decreases for  $x \leq 0$  in a small neighbourhood of  $x = 0$ . We may use sufficient conditions for optimality to demonstrate this fact. First of all, the first derivative of the function at  $x = 0$  is  $f'(x = 0) = 3x^2|_{x=0} = 0$ . Searching for a nonzero higher-order derivative, we observe that  $f''(x = 0) = 6x|_{x=0} = 0$  and  $f'''(x = 0) = 6|_{x=0} = 6$  (a nonzero number). Thus, the nonzero derivative occurs at  $n = 3$ , and since  $n$  is odd, the point  $x = 0$  is an inflection point.

### EXERCISE 2.1.2

Consider the optimality of the point  $x = 0$  in function  $f(x) = x^4$ . A plot of this function is shown in Figure 2.2. The point  $x = 0$  is a minimal point as can be seen from the figure. Since  $f'(x = 0) = 4x^3|_{x=0} = 0$ , we calculate higher-order derivatives in search of a nonzero derivative at  $x = 0$ :  $f''(0) = 0$ ,  $f'''(0) = 0$ , and  $f''''(0) = 24$ . Since the value of the fourth-order derivative is positive,  $n = 4$ , which is an even number. Thus the point  $x = 0$  is a local minimum point.

It is important to note here that even though a point can be tested for local optimality using the above conditions, global optimality of a point cannot be obtained using the above conditions. The common procedure for obtaining the global optimal point is to find a number of local optimal points tested using the above conditions and choose the best point. In Chapter 6, we discuss more about global optimization. In the following sections, we present a number of optimization techniques to find a local minimal point in a single-variable function.



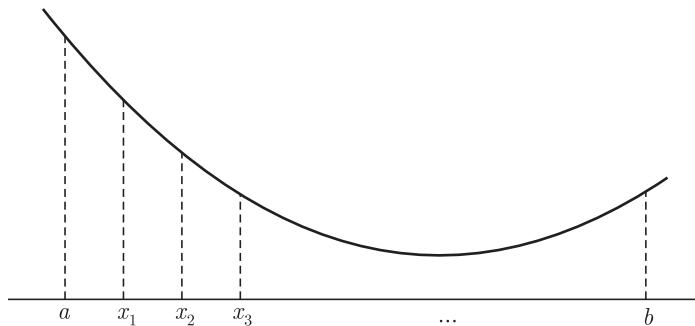
**Figure 2.2** The function  $f(x) = x^4$ .

## 2.2 Bracketing Methods

The minimum of a function is found in two phases. First, a crude technique is used to find a lower and an upper bound of the minimum. Thereafter, a more sophisticated method is used to search within these limits and find the optimal solution with the desired accuracy. In this section we discuss two methods for bracketing the minimum point and in the next section we shall discuss a number of methods to find the minimum point with the desired accuracy.

### 2.2.1 Exhaustive Search Method

We begin with the exhaustive search method, simply because this method is the simplest of all other methods. In the exhaustive search method, the optimum of a function is bracketed by calculating the function values at a number of equally spaced points (Figure 2.3). Usually, the search begins from a lower bound on the variable and three consecutive function values are compared at a time based on the assumption of unimodality of the function. Based on the outcome of comparison, the search is either terminated or continued by replacing one of the three points by a new point. The search continues until the minimum is bracketed.



**Figure 2.3** The exhaustive search method that uses equally spaced points.

### Algorithm

**Step 1** Set  $x_1 = a$ ,  $\Delta x = (b - a)/n$  ( $n$  is the number of intermediate points),  $x_2 = x_1 + \Delta x$ , and  $x_3 = x_2 + \Delta x$ .

**Step 2** If  $f(x_1) \geq f(x_2) \leq f(x_3)$ , the minimum point lies in  $(x_1, x_3)$ , **Terminate**;

Else  $x_1 = x_2$ ,  $x_2 = x_3$ ,  $x_3 = x_2 + \Delta x$ , and go to Step 3.

**Step 3** Is  $x_3 \leq b$ ? If yes, go to Step 2;

Else no minimum exists in  $(a, b)$  or a boundary point ( $a$  or  $b$ ) is the minimum point.

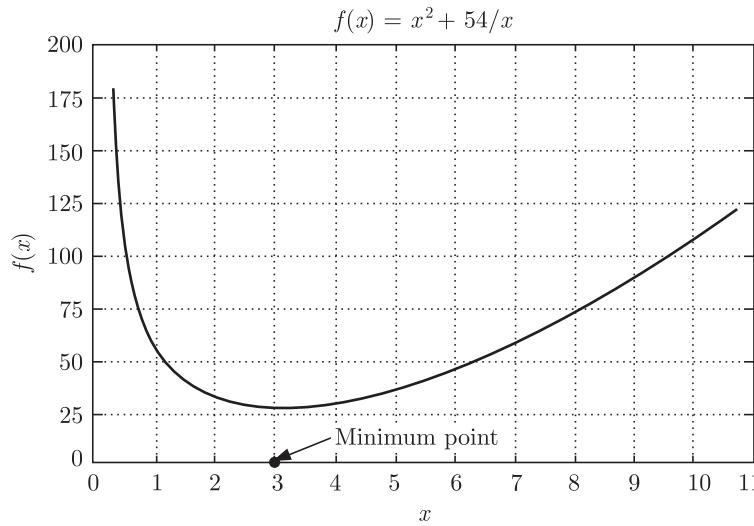
The final interval obtained by using this algorithm is  $2(b - a)/n$ . It can be computed that on an average  $(n/2 + 2)$  number of function evaluations are necessary to obtain the desired accuracy. To illustrate the working of the above algorithm, we consider the following minimization problem.

#### EXERCISE 2.2.1

Consider the problem:

$$\text{Minimize } f(x) = x^2 + 54/x$$

in the interval  $(0, 5)$ . A plot of the function is shown in Figure 2.4. The plot shows that the minimum lies at  $x^* = 3$ . The corresponding function value at



**Figure 2.4** The unimodal, single-variable function used in the exercise problems.  
The minimum point is at  $x = 3$ .

that point is  $f(x^*) = 27$ . By calculating the first and second derivatives at this point, we observe that  $f'(3) = 0$  and  $f''(3) = 6$ . Thus, the point  $x = 3$  is

a local minimum point, according to the sufficiency conditions for minimality described earlier.

In the following, we try to bracket the minimum of the above function using the exhaustive search method. Let us assume that we would like to bracket the minimum point by evaluating at most 11 different function values. Thus, we are considering only 10 intermediate points or  $n = 10$ .

**Step 1** According to the parameters chosen,  $x_1 = a = 0$  and  $b = 5$ . Since  $n = 10$ , the increment  $\Delta x = (5 - 0)/10$  or 0.5. We set  $x_2 = 0 + 0.5$  or 0.5 and  $x_3 = 0.5 + 0.5$  or 1.0.

**Step 2** Computing function values at various points, we have

$$f(0) = \infty, \quad f(0.5) = 108.25, \quad f(1.0) = 55.00.$$

Comparing these function values, we observe that  $f(x_1) > f(x_2) > f(x_3)$ . Thus, the minimum does not lie in the interval  $(0, 1)$ . We set  $x_1 = 0.5$ ,  $x_2 = 1.0$ ,  $x_3 = 1.5$ , and proceed to Step 3.

**Step 3** At this step, we observe that  $x_3 < 5$ . Therefore, we move to Step 2. This completes one iteration of the exhaustive search method. Since the minimum is not bracketed, we continue to perform the next iteration.

**Step 2** At this iteration, we have function values at  $x_1 = 0.5$ ,  $x_2 = 1.0$ , and  $x_3 = 1.5$ . Since we have already calculated function values at the first two points, we compute the function value at  $x_3$  only:  $f(x_3) = f(1.5) = 38.25$ . Thus,  $f(x_1) > f(x_2) > f(x_3)$ , and the minimum does not lie in the interval  $(0.5, 1.5)$ . Therefore, we set  $x_1 = 1.0$ ,  $x_2 = 1.5$ ,  $x_3 = 2.0$ , and move to Step 3.

**Step 3** Once again,  $x_3 < 5.0$ . Thus, we move to Step 2.

**Step 2** At this step, we require to compute the function value only at  $x_3 = 2.0$ . The corresponding function value is  $f(x_3) = f(2.0) = 31.00$ . Since  $f(x_1) > f(x_2) > f(x_3)$ , we continue with Step 3. We set  $x_1 = 1.5$ ,  $x_2 = 2.0$ , and  $x_3 = 2.5$ .

**Step 3** At this iteration,  $x_3 < 5.0$ . Thus, we move to Step 2.

**Step 2** The function value at  $x_3 = 2.5$  is  $f(x_3) = 27.85$ . Like previous iterations, we observe that  $f(x_1) > f(x_2) > f(x_3)$ , and therefore, we go to Step 3. The new set of three points are  $x_1 = 2.0$ ,  $x_2 = 2.5$ , and  $x_3 = 3.0$ . As evident from the progress of the algorithm so far that if the number of desired iterations to achieve the optimum is taken to be large to attain good accuracy in the solution, this method may lead to a number of iterations through Steps 2 and 3.

**Step 3** Once again,  $x_3 < 5.0$ . Thus, we move to Step 2.

**Step 2** Here,  $f(x_3) = f(3.0) = 27.00$ . Thus, we observe that  $f(x_1) > f(x_2) > f(x_3)$ . We set  $x_1 = 2.5$ ,  $x_2 = 3.0$ , and  $x_3 = 3.5$ .

**Step 3** Again,  $x_3 < 5.0$ , and we move to Step 2.

**Step 2** Here,  $f(x_3) = f(3.5) = 27.68$ . At this iteration we have a different situation:  $f(x_1) > f(x_2) < f(x_3)$ . This is precisely the condition for termination of the algorithm. Therefore, we have obtained a bound for the minimum:  $x^* \in (2.5, 3.5)$ .

As already stated, with  $n = 10$ , the accuracy of the solution can only be  $2(5 - 0)/10$  or 1.0, which is what we have obtained in the final interval. An interesting point to note is that if we require more precision in the obtained solution, we need to increase  $n$  or, in other words, we should be prepared to compute more function evaluations. For a desired accuracy in the solution, the parameter  $n$  has to be chosen accordingly. For example, if three decimal places of accuracy are desired in the above exercise problem, the required  $n$  can be obtained by solving the following equation for  $n$ :

$$\frac{2}{n}(b - a) = 0.001.$$

For  $a = 0$  and  $b = 5$ , about  $n = 10,000$  intermediate points are required and one an average,  $(n/2 + 2)$  or 5,002 function evaluations are required. In the above problem, with  $n = 10,000$ , the obtained interval is (2.9995, 3.0005).

### 2.2.2 Bounding Phase Method

Bounding phase method is used to bracket the minimum of a function. This method guarantees to bracket the minimum of a unimodal function. The algorithm begins with an initial guess and thereby finds a search direction based on two more function evaluations in the vicinity of the initial guess. Thereafter, an exponential search strategy is adopted to reach the optimum. In the following algorithm, an exponent of two is used, but any other value may very well be used.

#### Algorithm

**Step 1** Choose an initial guess  $x^{(0)}$  and an increment  $\Delta$ . Set  $k = 0$ .

**Step 2** If  $f(x^{(0)} - |\Delta|) \geq f(x^{(0)}) \geq f(x^{(0)} + |\Delta|)$ , then  $\Delta$  is positive;

Else if  $f(x^{(0)} - |\Delta|) \leq f(x^{(0)}) \leq f(x^{(0)} + |\Delta|)$ , then  $\Delta$  is negative;

Else go to Step 1.

**Step 3** Set  $x^{(k+1)} = x^{(k)} + 2^k \Delta$ .

**Step 4** If  $f(x^{(k+1)}) < f(x^{(k)})$ , set  $k = k + 1$  and go to Step 3;

Else the minimum lies in the interval  $(x^{(k-1)}, x^{(k+1)})$  and **Terminate**.

If the chosen  $\Delta$  is large, the bracketing accuracy of the minimum point is poor but the bracketing of the minimum is faster. On the other hand, if the chosen  $\Delta$  is small, the bracketing accuracy is better, but more function evaluations may be necessary to bracket the minimum. This method of bracketing the optimum is usually faster than exhaustive search method discussed in the previous section. We illustrate the working of this algorithm by taking the same exercise problem.

**EXERCISE 2.2.2**

We would like to bracket the minimum of the function

$$f(x) = x^2 + 54/x$$

using the bounding phase method.

**Step 1** We choose an initial guess  $x^{(0)} = 0.6$  and an increment  $\Delta = 0.5$ . We also set  $k = 0$ .

**Step 2** We calculate three function values to proceed with the algorithm:  $f(x^{(0)} - |\Delta|) = f(0.6 - 0.5) = 540.010$ ,  $f(x^{(0)}) = f(0.6) = 90.360$ , and  $f(x^{(0)} + |\Delta|) = f(0.6 + 0.5) = 50.301$ . We observe that  $f(0.1) > f(0.6) > f(1.1)$ . Thus we set  $\Delta = +0.5$ .

**Step 3** We compute the next guess:  $x^{(1)} = x^{(0)} + 2^0\Delta = 1.1$ .

**Step 4** The function value at  $x^{(1)}$  is 50.301 which is less than that at  $x^{(0)}$ . Thus, we set  $k = 1$  and go to Step 3. This completes one iteration of the bounding phase algorithm.

**Step 3** The next guess is  $x^{(2)} = x^{(1)} + 2^1\Delta = 1.1 + 2(0.5) = 2.1$ .

**Step 4** The function value at  $x^{(2)}$  is 30.124 which is smaller than that at  $x^{(1)}$ . Thus we set  $k = 2$  and move to Step 3.

**Step 3** We compute  $x^{(3)} = x^{(2)} + 2^2\Delta = 2.1 + 4(0.5) = 4.1$ .

**Step 4** The function value  $f(x^{(3)})$  is 29.981 which is smaller than  $f(x^{(2)}) = 31.124$ . We set  $k = 3$ .

**Step 3** The next guess is  $x^{(4)} = x^{(3)} + 2^3\Delta = 4.1 + 8(0.5) = 8.1$ .

**Step 4** The function value at this point is  $f(8.1) = 72.277$  which is larger than  $f(x^{(3)}) = 29.981$ . Thus, we terminate with the obtained interval as  $(2.1, 8.1)$ .

With  $\Delta = 0.5$ , the obtained bracketing is poor, but the number of function evaluations required is only 7. It is found that with  $x^{(0)} = 0.6$  and  $\Delta = 0.001$ , the obtained interval is  $(1.623, 4.695)$ , and the number of function evaluations is 15. The algorithm approaches the optimum exponentially but the accuracy in the obtained interval may not be very good, whereas in the exhaustive search method the iterations required to attain near the optimum may be large, but the obtained accuracy is good. An algorithm with a mixed strategy may be more desirable. At the end of this chapter, we present a FORTRAN code implementing this algorithm. A sample simulation result obtained using the code is also presented.

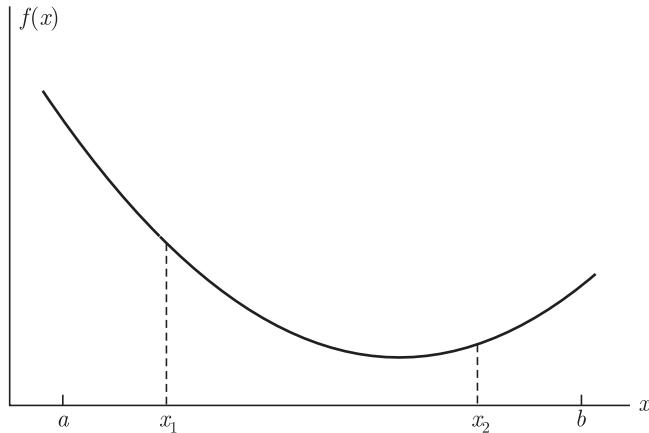
### 2.3 Region-Elimination Methods

Once the minimum point is bracketed, a more sophisticated algorithm needs to be used to improve the accuracy of the solution. In this section, we describe three algorithms that primarily work with the principle of region elimination and require comparatively smaller function evaluations. Depending on the function values evaluated at two points and assuming that the function is unimodal in the chosen search space, it can be concluded that the desired minimum cannot lie in some portion of the search space. The *fundamental rule* for region-elimination methods is as follows:

Let us consider two points  $x_1$  and  $x_2$  which lie in the interval  $(a, b)$  and satisfy  $x_1 < x_2$ . For unimodal functions for minimization, we can conclude the following:

- If  $f(x_1) > f(x_2)$  then the minimum does not lie in  $(a, x_1)$ .
- If  $f(x_1) < f(x_2)$  then the minimum does not lie in  $(x_2, b)$ .
- If  $f(x_1) = f(x_2)$  then the minimum does not lie in  $(a, x_1)$  and  $(x_2, b)$ .

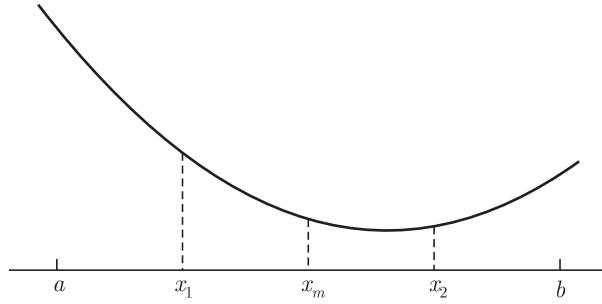
Consider a unimodal function drawn in Figure 2.5. If the function value at  $x_1$  is larger than that at  $x_2$ , the minimum point  $x^*$  cannot lie on the left-side of  $x_1$ . Thus, we can eliminate the region  $(a, x_1)$  from further consideration. Therefore, we reduce our interval of interest from  $(a, b)$  to  $(x_1, b)$ . Similarly, the second possibility ( $f(x_1) < f(x_2)$ ) can be explained. If the third situation occurs, that is, when  $f(x_1) = f(x_2)$  (this is a rare situation, especially when numerical computations are performed), we can conclude that regions  $(a, x_1)$  and  $(b, x_2)$  can be eliminated with the assumption that there exists only one local minimum in the search space  $(a, b)$ . The following algorithms constitute their search by using the above fundamental rule for region elimination.



**Figure 2.5** A typical single-variable unimodal function with function values at two distinct points.

### 2.3.1 Interval Halving Method

In this method, function values at three different points are considered. Three points divide the search space into four regions. Fundamental region elimination rule is used to eliminate a portion of the search space based on function values at the three chosen points. Three points chosen in the interval  $(a, b)$  are all equidistant from each other and equidistant from the boundaries by the same amount. Figure 2.6 shows these three points in the interval. Two of the function values are compared at a time and some region



**Figure 2.6** Three points  $x_1$ ,  $x_m$ , and  $x_2$  used in the interval halving method.

is eliminated. There are three scenarios that may occur. If  $f(x_1) < f(x_m)$ , then the minimum cannot lie beyond  $x_m$ . Therefore, we reduce the interval from  $(a, b)$  to  $(a, x_m)$ . The point  $x_m$  being the middle of the search space, this elimination reduces the search space to 50 per cent of the original search space. On the other hand, if  $f(x_1) > f(x_m)$ , the minimum cannot lie in the interval  $(a, x_1)$ . The point  $x_1$  being at one-fourth point in the search space, this reduction is only 25 per cent. Thereafter, we compare function values at  $x_m$  and  $x_2$  to eliminate further 25 per cent of the search space. This process continues until a small enough interval is found. The complete algorithm is described below. Since in each iteration of the algorithm, exactly half of the search space is retained, the algorithm is called the interval halving method.

#### Algorithm

**Step 1** Choose a lower bound  $a$  and an upper bound  $b$ . Choose also a small number  $\epsilon$ . Let  $x_m = (a + b)/2$ ,  $L_0 = L = b - a$ . Compute  $f(x_m)$ .

**Step 2** Set  $x_1 = a + L/4$ ,  $x_2 = b - L/4$ . Compute  $f(x_1)$  and  $f(x_2)$ .

**Step 3** If  $f(x_1) < f(x_m)$  set  $b = x_m$ ;  $x_m = x_1$ ; go to Step 5;  
Else go to Step 4.

**Step 4** If  $f(x_2) < f(x_m)$  set  $a = x_m$ ;  $x_m = x_2$ ; go to Step 5;  
Else set  $a = x_1$ ,  $b = x_2$ ; go to Step 5.

**Step 5** Calculate  $L = b - a$ . If  $|L| < \epsilon$ , **Terminate**;  
Else go to Step 2.

At every iteration, two new function evaluations are performed and the interval reduces to half of that at the previous iteration. Thus, the interval reduces to about  $0.5^{n/2}L_0$  after  $n$  function evaluations. Thus, the function evaluations required to achieve a desired accuracy  $\epsilon$  can be computed by solving the following equation:

$$(0.5)^{n/2}(b - a) = \epsilon.$$

### EXERCISE 2.3.1

We again consider the unimodal, single-variable function used before:

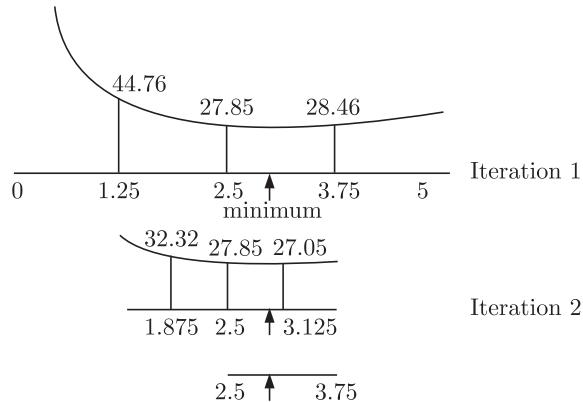
$$f(x) = x^2 + 54/x.$$

**Step 1** We choose  $a = 0$ ,  $b = 5$ , and  $\epsilon = 10^{-3}$ . The point  $x_m$  is the midpoint of the search interval. Thus,  $x_m = (0 + 5)/2 = 2.5$ . The initial interval length is  $L_0 = L = 5 - 0 = 5$ . The function value at  $x_m$  is  $f(x_m) = 27.85$ .

**Step 2** We set  $x_1 = 0 + 5/4 = 1.25$  and  $x_2 = 5 - 5/4 = 3.75$ . The corresponding function values are  $f(x_1) = 44.76$  and  $f(x_2) = 28.46$ .

**Step 3** By comparing these function values, we observe that  $f(x_1) > f(x_m)$ . Thus we continue with Step 4.

**Step 4** We again observe that  $f(x_2) > f(x_m)$ . Thus, we drop the intervals  $(0.00, 1.25)$  and  $(3.75, 5.00)$ . In other words, we set  $a = 1.25$  and  $b = 3.75$ . The outcome of this iteration is pictorially shown in Figure 2.7.



**Figure 2.7** First two iterations of the interval halving method. The figure shows how exactly half of the search space is eliminated at every iteration.

**Step 5** The new interval is  $L = 3.75 - 1.25 = 2.5$ , which is exactly half of that in the original interval ( $L_0 = 5$ ). Since  $|L|$  is not small, we continue with Step 2. This completes one iteration of the interval halving method.

**Step 2** We now compute new values of  $x_1$  and  $x_2$ :

$$x_1 = 1.25 + 2.5/4 = 1.875, \quad x_2 = 3.75 - 2.5/4 = 3.125.$$

The function values are  $f(x_1) = 32.32$  and  $f(x_2) = 27.05$ , respectively. It is important to note that even though three function values are required for comparison at Steps 3 and 4, we have to compute function values at two new points only; the other point ( $x_m$ , in this case) always happens to be one from the previous iteration.

**Step 3** We observe that  $f(x_1) = 32.32 > f(x_m) = 27.85$ . Thus, we go to Step 4.

**Step 4** Here,  $f(x_2) = 27.05 < f(x_m) = 27.85$ . Thus, we eliminate the interval  $(1.25, 2.5)$  and set  $a = 2.5$  and  $x_m = 3.125$ . This procedure is also depicted in Figure 2.7.

**Step 5** At the end of the second iteration, the new interval length is  $L = 3.75 - 2.5 = 1.25$ , which is again half of that in the previous iteration. Since this interval is not smaller than  $\epsilon$ , we perform another iteration.

**Step 2** We compute  $x_1 = 2.8125$  and  $x_2 = 3.4375$ . The corresponding function values are  $f(x_1) = 27.11$  and  $f(x_2) = 27.53$ .

**Step 3** We observe that  $f(x_1) = 27.11 > f(x_m) = 27.05$ . So we move to Step 4.

**Step 4** Here,  $f(x_2) = 27.53 > f(x_m) = 27.05$  and we drop the boundary intervals. Thus,  $a = 2.8125$  and  $b = 3.4375$ .

**Step 5** The new interval  $L = 0.625$ . We continue this process until an  $L$  smaller than a specified small value ( $\epsilon$ ) is obtained.

We observe that at the end of each iteration, the interval is reduced to half of its original size and after three iterations, the interval is  $(\frac{1}{2})^3 L_0 = 0.625$ . Since two function evaluations are required per iteration and half of the region is eliminated at each iteration, the effective region elimination per function evaluation is 25 per cent. In the following subsections, we discuss two more algorithms with larger region elimination capabilities per function evaluation.

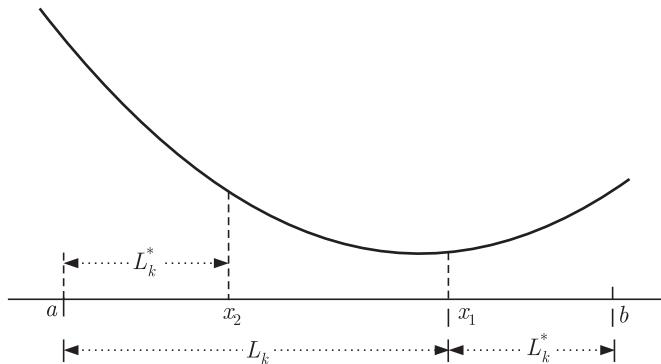
### 2.3.2 Fibonacci Search Method

In this method, the search interval is reduced according to Fibonacci numbers. The property of the Fibonacci numbers is that, given two consecutive numbers  $F_{n-2}$  and  $F_{n-1}$ , the third number is calculated as follows:

$$F_n = F_{n-1} + F_{n-2}, \tag{2.1}$$

where  $n = 2, 3, 4, \dots$ . The first few Fibonacci numbers are  $F_0 = 1$ ,  $F_1 = 1$ ,  $F_2 = 2$ ,  $F_3 = 3$ ,  $F_4 = 5$ ,  $F_5 = 8$ ,  $F_6 = 13$ , and so on. The property of the Fibonacci numbers can be used to create a search algorithm that requires

only one function evaluation at each iteration. The principle of Fibonacci search is that out of two points required for the use of the region-elimination rule, one is always the previous point and the other point is new. Thus, only one function evaluation is required at each iteration. At iteration  $k$ , two intermediate points, each  $L_k^*$  away from either end of the search space ( $L = b - a$ ) are chosen. When the region-elimination rule eliminates a portion of the search space depending on the function values at these two points, the remaining search space is  $L_k$ . By defining  $L_k^* = (F_{n-k+1}/F_{n+1})L$  and  $L_k = (F_{n-k+2}/F_{n+1})L$ , it can be shown that  $L_k - L_k^* = L_{k+1}^*$ , which means that one of the two points used in iteration  $k$  remains as one point in iteration  $(k + 1)$ . This can be seen from Figure 2.8. If the region  $(a, x_2)$  is eliminated in the  $k$ -th iteration, the point  $x_1$  is at a distance  $(L_k - L_k^*)$  or  $L_{k+1}^*$  from the point  $x_2$  in the  $(k + 1)$ -th iteration. Since, the first two Fibonacci numbers are the same, the algorithm usually starts with  $k = 2$ .



**Figure 2.8** Fibonacci search points ( $x_1$  and  $x_2$ ).

### Algorithm

**Step 1** Choose a lower bound  $a$  and an upper bound  $b$ . Set  $L = b - a$ . Assume the desired number of function evaluations to be  $n$ . Set  $k = 2$ .

**Step 2** Compute  $L_k^* = (F_{n-k+1}/F_{n+1})L$ . Set  $x_1 = a + L_k^*$  and  $x_2 = b - L_k^*$ .

**Step 3** Compute one of  $f(x_1)$  or  $f(x_2)$ , which was not evaluated earlier. Use the fundamental region-elimination rule to eliminate a region. Set new  $a$  and  $b$ .

**Step 4** Is  $k = n$ ? If no, set  $k = k + 1$  and go to Step 2;

Else **Terminate**.

In this algorithm, the interval reduces to  $(2/F_{n+1})L$  after  $n$  function evaluations. Thus, for a desired accuracy  $\epsilon$ , the number of required function evaluations  $n$  can be calculated using the following equation:

$$\frac{2}{F_{n+1}}(b - a) = \epsilon.$$

As is clear from the algorithm, only one function evaluation is required at each iteration. At iteration  $k$ , a proportion of  $F_{n-k}/F_{n-k+2}$  of the search space at the previous iteration is eliminated. For large values of  $n$ , this quantity is close to 38.2 per cent, which is better than that in the interval halving method. (Recall that in the interval halving method this quantity is 25 per cent.) However, one difficulty with this algorithm is that the Fibonacci numbers must be calculated in each iteration.

We illustrate the working of this algorithm on the same function used earlier.

### EXERCISE 2.3.2

Minimize the function

$$f(x) = x^2 + 54/x.$$

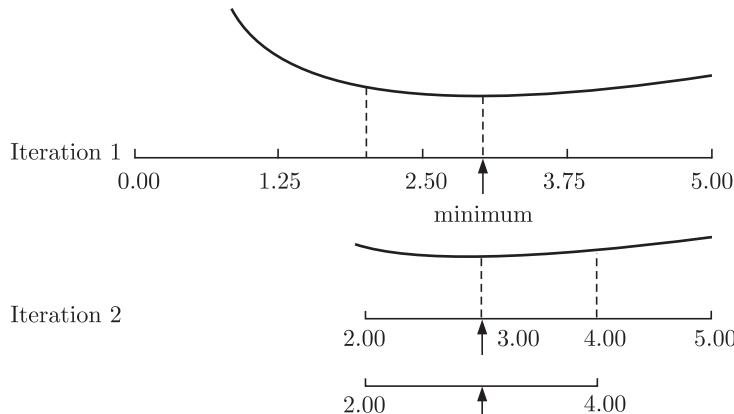
**Step 1** We choose  $a = 0$  and  $b = 5$ . Thus, the initial interval is  $L = 5$ . Let us also choose the desired number of function evaluations to be three ( $n = 3$ ). In practice, a large value of  $n$  is usually chosen. We set  $k = 2$ .

**Step 2** We compute  $L_2^*$  as follows:

$$L_2^* = (F_{3-2+1}/F_{3+1})L = (F_2/F_4) \cdot 5 = \frac{2}{5} \cdot 5 = 2.$$

Thus, we calculate  $x_1 = 0 + 2 = 2$  and  $x_2 = 5 - 2 = 3$ .

**Step 3** We compute the function values:  $f(x_1) = 31$  and  $f(x_2) = 27$ . Since  $f(x_1) > f(x_2)$ , we eliminate the region  $(0, x_1)$  or  $(0, 2)$ . In other words, we set  $a = 2$  and  $b = 5$ . Figure 2.9 shows the function values at these two points and the resulting region after Step 3. The exact minimum of the function is also shown.



**Figure 2.9** Two iterations of the Fibonacci search method.

**Step 4** Since  $k = 2 \neq n = 3$ , we set  $k = 3$  and go to Step 2. This completes one iteration of the Fibonacci search method.

**Step 2** We compute  $L_3^* = (F_1/F_4)L = \frac{1}{5} \cdot 5 = 1$ ,  $x_1 = 2 + 1 = 3$ , and  $x_2 = 5 - 1 = 4$ .

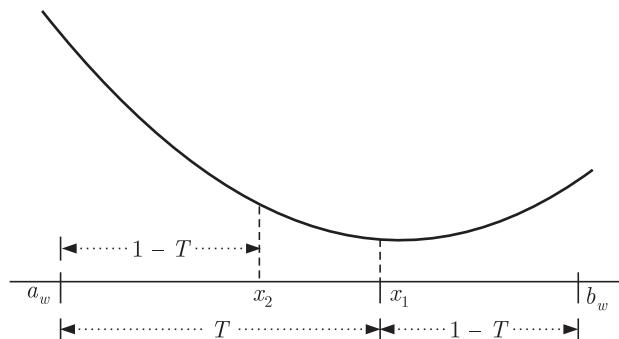
**Step 3** We observe that one of the points ( $x_1 = 3$ ) was evaluated in the previous iteration. It is important to note that this is not an accident. The property of the Fibonacci search method is such that at every iteration only one new point will be considered. Thus, we need to compute the function value only at point  $x_2 = 4$ :  $f(x_2) = 29.5$ . By comparing function values at  $x_1 = 3$  and  $x_2 = 4$ , we observe that  $f(x_1) < f(x_2)$ . Therefore, we set  $a = 2$  and  $b = x_2 = 4$ , since the fundamental rule suggests that the minimum cannot lie beyond  $x_2 = 4$ .

**Step 4** At this iteration,  $k = n = 3$  and we terminate the algorithm. The final interval is  $(2, 4)$ .

As already stated, after three function evaluations, the interval reduces to  $(2/F_4)L$  or  $(\frac{2}{5} \times 5)$  or 2. The progress of the above two iterations is shown in Figure 2.9. For a better accuracy, a larger value of  $n$  is required to be set and more iterations will be required to achieve that accuracy.

### 2.3.3 Golden Section Search Method

One difficulty of the Fibonacci search method is that the Fibonacci numbers have to be calculated and stored. Another problem is that at every iteration the proportion of the eliminated region is not the same. In order to overcome these two problems and yet calculate one new function evaluation per iteration, the golden section search method is used. In this algorithm, the search space  $(a, b)$  is first linearly mapped to a unit interval search space  $(0, 1)$ . Thereafter, two points at  $\tau$  from either end of the search space are chosen so that at every iteration the eliminated region is  $(1 - \tau)$  to that in the previous iteration (Figure 2.10). This can be achieved by equating  $1 - \tau$  with  $(\tau \times \tau)$ . This yields the golden number:  $\tau = 0.618$ . Figure 2.10 can be used to verify that in each iteration one of the two points  $x_1$  and  $x_2$  is always a point considered in the previous iteration.



**Figure 2.10** The points  $(x_1$  and  $x_2)$  used in the golden section search method.

### Algorithm

**Step 1** Choose a lower bound  $a$  and an upper bound  $b$ . Also choose a small number  $\epsilon$ . Normalize the variable  $x$  by using the equation  $w = (x - a)/(b - a)$ . Thus,  $a_w = 0$ ,  $b_w = 1$ , and  $L_w = 1$ . Set  $k = 1$ .

**Step 2** Set  $w_1 = a_w + (0.618)L_w$  and  $w_2 = b_w - (0.618)L_w$ . Compute  $f(w_1)$  or  $f(w_2)$ , depending on whichever of the two was not evaluated earlier. Use the fundamental region-elimination rule to eliminate a region. Set new  $a_w$  and  $b_w$ .

**Step 3** Is  $|L_w| < \epsilon$  small? If no, set  $k = k + 1$ , go to Step 2;

Else **Terminate**.

In this algorithm, the interval reduces to  $(0.618)^{n-1}$  after  $n$  function evaluations. Thus, the number of function evaluations  $n$  required to achieve a desired accuracy  $\epsilon$  is calculated by solving the following equation:

$$(0.618)^{n-1}(b - a) = \epsilon.$$

Like the Fibonacci method, only one function evaluation is required at each iteration and the effective region elimination per function evaluation is exactly 38.2 per cent, which is higher than that in the interval halving method. This quantity is the same as that in the Fibonacci search for large  $n$ . In fact, for a large  $n$ , the Fibonacci search is equivalent to the golden section search.

### EXERCISE 2.3.3

Consider the following function again:

$$f(x) = x^2 + 54/x.$$

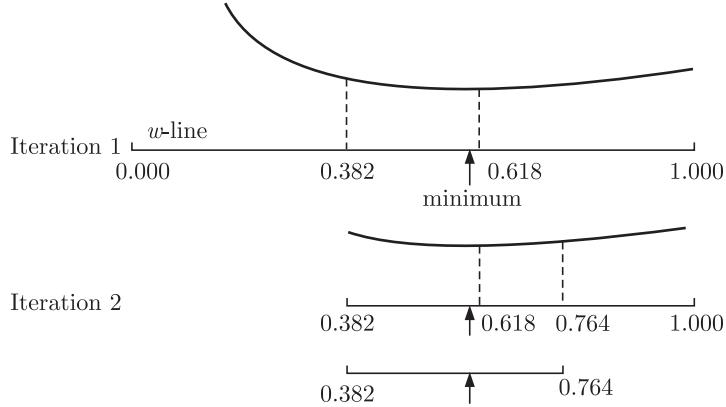
**Step 1** We choose  $a = 0$  and  $b = 5$ . The transformation equation becomes  $w = x/5$ . Thus,  $a_w = 0$ ,  $b_w = 1$ , and  $L_w = 1$ . Since the golden section method works with a transformed variable  $w$ , it is convenient to work with the transformed function:

$$f(w) = 25w^2 + 54/(5w).$$

In the  $w$ -space, the minimum lies at  $w^* = 3/5 = 0.6$ . We set an iteration counter  $k = 1$ .

**Step 2** We set  $w_1 = 0 + (0.618)1 = 0.618$  and  $w_2 = 1 - (0.618)1$  or  $w_2 = 0.382$ . The corresponding function values are  $f(w_1) = 27.02$  and  $f(w_2) = 31.92$ . Since  $f(w_1) < f(w_2)$ , the minimum cannot lie in any point smaller than  $w = 0.382$ . Thus, we eliminate the region  $(a, w_2)$  or  $(0, 0.382)$ . Thus,  $a_w = 0.382$  and  $b_w = 1$ . At this stage,  $L_w = 1 - 0.382 = 0.618$ . The region being eliminated after this iteration is shown in Figure 2.11. The position of the exact minimum at  $w = 0.6$  is also shown.

**Step 3** Since  $|L_w|$  is not smaller than  $\epsilon$ , we set  $k = 2$  and move to Step 2. This completes one iteration of the golden section search method.



**Figure 2.11** Region eliminations in the first two iterations of the golden section search algorithm.

**Step 2** For the second iteration, we set

$$w_1 = 0.382 + (0.618)0.618 = 0.764,$$

$$w_2 = 1 - (0.618)0.618 = 0.618.$$

We observe that the point  $w_2$  was computed in the previous iteration. Thus, we only need to compute the function value at  $w_1$ :  $f(w_1) = 28.73$ . Using the fundamental region-elimination rule and observing the relation  $f(w_1) > f(w_2)$ , we eliminate the interval  $(0.764, 1)$ . Thus, the new bounds are  $a_w = 0.382$  and  $b_w = 0.764$ , and the new interval is  $L_w = 0.764 - 0.382 = 0.382$ , which is incidentally equal to  $(0.618)^2$ ! Figure 2.11 shows the final region after two iterations of this algorithm.

**Step 3** Since the obtained interval is not smaller than  $\epsilon$ , we continue to proceed to Step 2 after incrementing the iteration counter  $k$  to 3.

**Step 2** Here, we observe that  $w_1 = 0.618$  and  $w_2 = 0.528$ , of which the point  $w_1$  was evaluated before. Thus, we compute  $f(w_2)$  only:  $f(w_2) = 27.43$ . We also observe that  $f(w_1) < f(w_2)$  and we eliminate the interval  $(0.382, 0.528)$ . The new interval is  $(0.528, 0.764)$  and the new range is  $L_w = 0.764 - 0.528 = 0.236$ , which is exactly equal to  $(0.618)^3$ !

**Step 3** Thus, at the end of the third iteration,  $L_w = 0.236$ . This way, Steps 2 and 3 may be continued until the desired accuracy is achieved.

We observe that at each iteration, only one new function evaluation is necessary. After three iterations, we have performed only four function evaluations. Thus, the interval reduces to  $(0.618)^3$  or 0.236.

At the end of this chapter, we present a FORTRAN code implementing this algorithm. A simulation run on the above function obtained using this

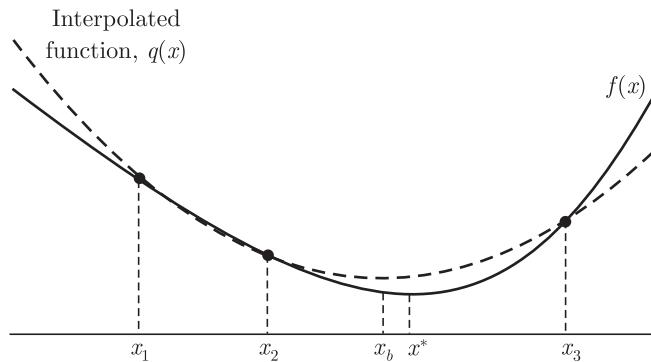
algorithm is also presented. For other functions, the subroutine `funct` may be modified and rerun the code.

## 2.4 Point-Estimation Method

In the previous search methods, only relative function values of two points were considered to guide the search, but the magnitude of the function values at the chosen points may also provide some information about the location of the minimum in the search space. The successive quadratic estimation method described below represents a class of point-estimation methods which use the magnitude and sign of function values to help guide the search. Typically, the function values are computed at a number of points and a unimodal function is fitted through these points exactly. The minimum of the fitted function is considered to be a guess of the minimum of the original objective function.

### 2.4.1 Successive Quadratic Estimation Method

In this algorithm, the fitted curve is a quadratic polynomial function. Since any quadratic function can be defined with three points, the algorithm begins with three initial points. Figure 2.12 shows the original function and three



**Figure 2.12** The function  $f(x)$  and the interpolated quadratic function.

initial points  $x_1$ ,  $x_2$ , and  $x_3$ . The fitted quadratic curve through these three points is also plotted with a dashed line. The minimum ( $\bar{x}$ ) of this curve is used as one of the candidate points for the next iteration. For non-quadratic functions, a number of iterations of this algorithm is necessary, whereas for quadratic objective functions the exact minimum can be found in one iteration only.

A general quadratic function passing through two points  $x_1$  and  $x_2$  can be written as

$$q(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2).$$

If  $(x_1, f_1)$ ,  $(x_2, f_2)$ , and  $(x_3, f_3)$  are three points on this function, then the following relationships can be obtained:

$$a_0 = f_1, \quad (2.2)$$

$$a_1 = \frac{f_2 - f_1}{x_2 - x_1}, \quad (2.3)$$

$$a_2 = \frac{1}{x_3 - x_2} \left( \frac{f_3 - f_1}{x_3 - x_1} - a_1 \right). \quad (2.4)$$

By differentiating  $q(x)$  with respect to  $x$  and setting it to zero, it can be shown that the minimum of the above function is

$$\bar{x} = \frac{x_1 + x_2}{2} - \frac{a_1}{2a_2}. \quad (2.5)$$

The above point is an estimate of the minimum point provided  $q''(\bar{x}) > 0$  or  $a_2 > 0$ , which depends only on the choice of the three basic points. Among the four points ( $x_1$ ,  $x_2$ ,  $x_3$ , and  $\bar{x}$ ), the best three points are kept and a new interpolated function  $q(x)$  is found again. This procedure continues until two consecutive estimates are close to each other.

Based on these results, we present Powell's algorithm (Powell, 1964).

### Algorithm

**Step 1** Let  $x_1$  be an initial point and  $\Delta$  be the step size. Compute  $x_2 = x_1 + \Delta$ .

**Step 2** Evaluate  $f(x_1)$  and  $f(x_2)$ .

**Step 3** If  $f(x_1) > f(x_2)$ , let  $x_3 = x_1 + 2\Delta$ ; Else let  $x_3 = x_1 - \Delta$ . Evaluate  $f(x_3)$ .

**Step 4** Determine  $F_{\min} = \min(f_1, f_2, f_3)$  and  $X_{\min}$  is the point  $x_i$  that corresponds to  $F_{\min}$ .

**Step 5** Use points  $x_1$ ,  $x_2$ , and  $x_3$  to calculate  $\bar{x}$  using Equation (2.5).

**Step 6** Are  $|F_{\min} - f(\bar{x})|$  and  $|X_{\min} - \bar{x}|$  small? If not, go to Step 7; Else the optimum is the best of current four points and **Terminate**.

**Step 7** Save the best point and two bracketing it, if possible; otherwise, save the best three points. Relabel them according to  $x_1 < x_2 < x_3$  and go to Step 4.

In the above algorithm, no check is made to satisfy  $a_2 > 0$ . The same can be incorporated in Step 5. If  $a_2$  is found to be negative, one of the three points may be replaced by a random point. This process is continued until the quantity  $a_2$  becomes nonnegative.

**EXERCISE 2.4.1**

We consider again the same unimodal, single-variable function

$$f(x) = x^2 + 54/x$$

to illustrate the working principle of the algorithm.

**Step 1** We choose  $x_1 = 1$  and  $\Delta = 1$ . Thus,  $x_2 = 1 + 1 = 2$ .

**Step 2** The corresponding function values are  $f(x_1) = 55$  and  $f(x_2) = 31$ .

**Step 3** Since  $f(x_1) > f(x_2)$ , we set  $x_3 = 1 + 2(1) = 3$  and the function value is  $f(x_3) = 27$ .

**Step 4** By comparing function values at these points, we observe that the minimum function value  $F_{\min} = \min(55, 31, 27) = 27$  and the corresponding point is  $X_{\min} = x_3 = 3$ .

**Step 5** Using Equations (2.2) to (2.4), we calculate the following parameters:

$$a_0 = 55,$$

$$a_1 = \frac{31 - 55}{2 - 1} = -24,$$

$$a_2 = \frac{1}{3 - 2} \left[ \frac{27 - 55}{3 - 1} - (-24) \right] = 10.$$

Since  $a_2 > 0$ , the estimated minimum is

$$\bar{x} = (1 + 2)/2 - (-24)/(2 \times 10) = 2.7.$$

The corresponding function value is  $f(\bar{x}) = 27.29$ .

**Step 6** Let us assume that  $|27 - 27.29|$  and  $|3 - 2.7|$  are not small enough to terminate. Thus, we proceed to Step 7.

**Step 7** The best point is  $x_3 = 3$ , which is an extreme point. Thus, we consider the best three points:  $x_1 = 2$ ,  $x_2 = 2.7$ , and  $x_3 = 3$ . This completes one iteration of the algorithm. To continue with the next iteration, we proceed to Step 4.

**Step 4** At this stage,  $F_{\min} = \min(31, 27.29, 27) = 27$  and the corresponding point is  $X_{\min} = 3$ .

**Step 5** Using Equation (2.2), we obtain  $a_1 = -5.3$  and  $a_2 = 4.33$ , which is positive. The estimated minimum is  $\bar{x} = 2.96$ . The corresponding function value is  $f(\bar{x}) = 27.005$ .

**Step 6** Here, the values  $|27 - 27.005|$  and  $|3 - 2.96|$  may be assumed to be small. Therefore, we terminate the process and declare that the minimum solution of the function is the best of current four points. In this case, the minimum is  $x^* = 3$  with  $f(x^*) = 27$ .

It is observed that for well-behaved unimodal functions, this method finds the minimum point faster than the region-elimination methods. But for skewed functions, the golden section search is better than Powell's method.

## 2.5 Gradient-based Methods

We have demonstrated that all the methods described in earlier sections work with direct function values and not with derivative information. The algorithms discussed in this section require derivative information. In many real-world problems, it is difficult to obtain the information about derivatives, either due to the nature of the problem or due to the computations involved in calculating the derivatives. Despite these difficulties, gradient-based methods are popular and are often found to be effective. However, it is recommended to use these algorithms in problems where the derivative information is available or can be calculated easily. The optimality property that at a local or a global optimum the gradient is zero can be used to terminate the search process.

### 2.5.1 Newton-Raphson Method

The goal of an unconstrained local optimization method is to achieve a point having as small a derivative as possible. In the Newton-Raphson method, a linear approximation to the first derivative of the function is made at a point using the Taylor's series expansion. That expression is equated to zero to find the next guess. If the current point at iteration  $t$  is  $x^{(t)}$ , the point in the next iteration is governed by the following simple equation (obtained by considering up to the linear term in Taylor's series expansion):

$$x^{(t+1)} = x^{(t)} - \frac{f'(x^{(t)})}{f''(x^{(t)})}. \quad (2.6)$$

#### Algorithm

**Step 1** Choose initial guess  $x^{(1)}$  and a small number  $\epsilon$ . Set  $k = 1$ . Compute  $f'(x^{(1)})$ .

**Step 2** Compute  $f''(x^{(k)})$ .

**Step 3** Calculate  $x^{(k+1)} = x^{(k)} - f'(x^{(k)})/f''(x^{(k)})$ . Compute  $f'(x^{(k+1)})$ .

**Step 4** If  $|f'(x^{(k+1)})| < \epsilon$ , **Terminate**;

Else set  $k = k + 1$  and go to Step 2.

Convergence of the algorithm depends on the initial point and the nature of the objective function. For mathematical functions, the derivative may be easy to compute, but in practice, the gradients have to be computed

numerically. At a point  $x^{(t)}$ , the first and second derivatives are computed as follows, using the central difference method (Scarborough, 1966):

$$f'(x^{(t)}) = \frac{f(x^{(t)} + \Delta x^{(t)}) - f(x^{(t)} - \Delta x^{(t)})}{2\Delta x^{(t)}}, \quad (2.7)$$

$$f''(x^{(t)}) = \frac{f(x^{(t)} + \Delta x^{(t)}) - 2f(x^{(t)}) + f(x^{(t)} - \Delta x^{(t)})}{(\Delta x^{(t)})^2}. \quad (2.8)$$

The parameter  $\Delta x^{(t)}$  is usually taken to be a small value. In all our calculations, we assign  $\Delta x^{(t)}$  to be about 1 per cent of  $x^{(t)}$ :

$$\Delta x^{(t)} = \begin{cases} 0.01|x^{(t)}|, & \text{if } |x^{(t)}| > 0.01, \\ 0.0001, & \text{otherwise.} \end{cases} \quad (2.9)$$

According to Equations (2.7) and (2.8), the first derivative requires two function evaluations and the second derivative requires three function evaluations.

### EXERCISE 2.5.1

Consider the minimization problem:

$$f(x) = x^2 + 54/x.$$

**Step 1** We choose an initial guess  $x^{(1)} = 1$ , a termination factor  $\epsilon = 10^{-3}$ , and an iteration counter  $k = 1$ . We compute the derivative using Equation (2.7). The small increment as computed using Equation (2.9) is 0.01. The computed derivative is  $-52.005$ , whereas the exact derivative at  $x^{(1)}$  is found to be  $-52$ . We accept the computed derivative value and proceed to Step 2.

**Step 2** The exact second derivative of the function at  $x^{(1)} = 1$  is found to be 110. The second derivative computed using Equation (2.8) is  $f''(x^{(1)}) = 110.011$ , which is close to the exact value.

**Step 3** We compute the next guess,

$$\begin{aligned} x^{(2)} &= x^{(1)} - f'(x^{(1)})/f''(x^{(1)}), \\ &= 1 - (-52.005)/(110.011), \\ &= 1.473. \end{aligned}$$

The derivative computed using Equation (2.7) at this point is found to be  $f'(x^{(2)}) = -21.944$ .

**Step 4** Since  $|f'(x^{(2)})| \not< \epsilon$ , we increment  $k$  to 2 and go to Step 2. This completes one iteration of the Newton-Raphson method.

**Step 2** We begin the second iteration by computing the second derivative numerically at  $x^{(2)}$ :  $f''(x^{(2)}) = 35.796$ .

**Step 3** The next guess, as computed using Equation (2.6), is  $x^{(3)} = 2.086$  and  $f'(x^{(3)}) = -8.239$  computed numerically.

**Step 4** Since  $|f'(x^{(3)})| \not< \epsilon$ , we set  $k = 3$  and move to Step 2. This is the end of the second iteration.

**Step 2** The second derivative at the point is  $f''(x^{(3)}) = 13.899$ .

**Step 3** The new point is calculated as  $x^{(4)} = 2.679$  and the derivative is  $f'(x^{(4)}) = -2.167$ . Nine function evaluations were required to obtain this point.

**Step 4** Since the absolute value of this derivative is not smaller than  $\epsilon$ , the search proceeds to Step 2.

After three more iterations, we find that  $x^{(7)} = 3.0001$  and the derivative is  $f'(x^{(7)}) = -4(10)^{-8}$ , which is small enough to terminate the algorithm. Since, at every iteration the first and second-order derivatives are calculated at a new point, a total of three function values are evaluated at every iteration.

### 2.5.2 Bisection Method

The Newton-Raphson method involves computation of the second derivative, a numerical computation of which requires three function evaluations. In the bisection method, the computation of the second derivative is avoided; instead, only the first derivative is used. Both the function value and the sign of the first derivative at two points is used to eliminate a certain portion of the search space. This method is similar to the region-elimination methods discussed in Section 2.3.1, but in this method, derivatives are used to make the decision about the region to be eliminated. The algorithm once again assumes the unimodality of the function.

Using the derivative information, the minimum is said to be bracketed in the interval  $(a, b)$  if two conditions— $f'(a) < 0$  and  $f'(b) > 0$ —are satisfied. Like other region-elimination methods, this algorithm also requires two initial boundary points bracketing the minimum. A bracketing algorithm described in Section 2.2 may be used to find the bracketing points. In the bisection method, derivatives at two boundary points and at the middle point are calculated and compared. Of the three points, two consecutive points with derivatives having opposite signs are chosen for the next iteration.

#### Algorithm

**Step 1** Choose two points  $a$  and  $b$  such that  $f'(a) < 0$  and  $f'(b) > 0$ . Also choose a small number  $\epsilon$ . Set  $x_1 = a$  and  $x_2 = b$ .

**Step 2** Calculate  $z = (x_2 + x_1)/2$  and evaluate  $f'(z)$ .

**Step 3** If  $|f'(z)| \leq \epsilon$ , **Terminate**;  
 Else if  $f'(z) < 0$  set  $x_1 = z$  and go to Step 2;  
 Else if  $f'(z) > 0$  set  $x_2 = z$  and go to Step 2.

The sign of the first-derivative at the mid-point of the current search region is used to eliminate half of the search region. If the derivative is negative, the minimum cannot lie in the left-half of the search region and if the derivative is positive, the minimum cannot lie in the right-half of the search space.

### EXERCISE 2.5.2

Consider again the function:

$$f(x) = x^2 + 54/x.$$

**Step 1** We choose two points  $a = 2$  and  $b = 5$  such that  $f'(a) = -9.501$  and  $f'(b) = 7.841$  are of opposite sign. The derivatives are computed numerically using Equation (2.7). We also choose a small number  $\epsilon = 10^{-3}$ .

**Step 2** We calculate a quantity  $z = (x_1 + x_2)/2 = 3.5$  and compute  $f'(z) = 2.591$ .

**Step 3** Since  $f'(z) > 0$ , the right-half of the search space needs to be eliminated. Thus, we set  $x_1 = 2$  and  $x_2 = z = 3.5$ . This completes one iteration of the algorithm. This algorithm works more like the interval halving method described in Section 2.4. At each iteration, only half of the search region is eliminated, but here the decision about which half to delete depends on the derivatives at the mid-point of the interval.

**Step 2** We compute  $z = (2 + 3.5)/2 = 2.750$  and  $f'(z) = -1.641$ .

**Step 3** Since  $f'(z) < 0$ , we set  $x_1 = 2.750$  and  $x_2 = 3.500$ .

**Step 2** The new point  $z$  is the average of the two bounds:  $z = 3.125$ . The function value at this point is  $f'(z) = 0.720$ .

**Step 3** Since  $|f'(z)| \not\leq \epsilon$ , we continue with Step 2.

Thus, at the end of 10 function evaluations, we have obtained an interval  $(2.750, 3.125)$ , bracketing the minimum point  $x^* = 3.0$ . The guess of the minimum point is the mid-point of the obtained interval or  $x = 2.938$ . This process continues until we find a point with a vanishing derivative. Since at each iteration, the gradient is evaluated only at one new point, the bisection method requires two function evaluations per iteration. In this method, exactly half the region is eliminated at every iteration; but using the magnitude of the gradient, a faster algorithm can be designed to adaptively eliminate variable portions of search region—a matter which we discuss in the following subsection.

### 2.5.3 Secant Method

In the secant method, both magnitude and sign of derivatives are used to create a new point. The derivative of the function is assumed to vary linearly between the two chosen boundary points. Since boundary points have derivatives with opposite signs and the derivatives vary linearly between the boundary points, there exists a point between these two points with a zero derivative. Knowing the derivatives at the boundary points, the point with zero derivative can be easily found. If at two points  $x_1$  and  $x_2$ , the quantity  $f'(x_1)f'(x_2) \leq 0$ , the linear approximation of the derivative  $x_1$  and  $x_2$  will have a zero derivative at the point  $z$  given by

$$z = x_2 - \frac{f'(x_2)}{(f'(x_2) - f'(x_1))/(x_2 - x_1)}. \quad (2.10)$$

In this method, in one iteration more than half the search space may be eliminated depending on the gradient values at the two chosen points. However, smaller than half the search space may also be eliminated in one iteration.

#### Algorithm

The algorithm is the same as the bisection method except that Step 2 is modified as follows:

**Step 2** Calculate the new point  $z$  using Equation (2.10) and evaluate  $f'(z)$ .

This algorithm also requires only one gradient evaluation at every iteration. Thus, only two function values are required per iteration.

#### EXERCISE 2.5.3

Consider once again the function:

$$f(x) = x^2 + 54/x.$$

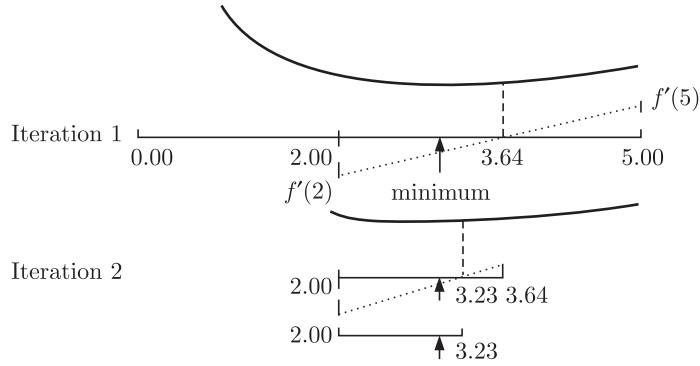
**Step 1** We begin with initial points  $a = 2$  and  $b = 5$  having derivatives  $f'(a) = -9.501$  and  $f'(b) = 7.841$  with opposite signs. We also choose  $\epsilon = 10^{-3}$  and set  $x_1 = 2$  and  $x_2 = 5$ . In any problem, a number of iterations may be required to find two points with opposite gradient values.

**Step 2** We now calculate a new point using Equation (2.10):

$$z = 5 - f'(5) \left/ \left( \frac{(f'(5) - f'(2))}{(5 - 2)} \right) \right. = 3.644.$$

The derivative at this point, computed numerically, is  $f'(z) = 3.221$ . The mechanism of finding the new point is depicted in Figure 2.13.

**Step 3** Since  $f'(z) > 0$ , we eliminate the right part (the region  $(z, b)$ ) of the original search region. The amount of eliminated search space is



**Figure 2.13** Two iterations of the region-elimination technique in the secant method.

$(b - z) = 1.356$ , which is less than half the search space  $(b - a)/2 = 2.5$ . We set  $x_1 = 2$  and  $x_2 = 3.644$ . This completes one iteration of the secant method.

**Step 2** The next point which is computed using Equation (2.10) is  $z = 3.228$ . The derivative at this point is  $f'(z) = 1.127$ .

**Step 3** Since  $f'(z) > 0$ , we eliminate the right part of the search space, that is, we discard the region  $(3.228, 3.644)$ . The amount of eliminated search space is 0.416, which is also smaller than half of the previous search space  $(3.644 - 2)/2$  or 0.822. In both these iterations, the eliminated region is less than half of the search space, but in some iterations, a region more than the half of the search space can also be eliminated. Thus, we set  $x_1 = 2$  and  $x_2 = 3.228$ .

**Step 2** The new point,  $z = 3.101$  and  $f'(z) = 0.586$ .

**Step 3** Since  $|f'(z)| \not< \epsilon$ , we continue with Step 2.

At the end of 10 function evaluations, the guess of the true minimum point is computed using Equation (2.10):  $x = 3.037$ . This point is closer to the true minimum point ( $x^* = 3.0$ ) than that obtained using the bisection method.

#### 2.5.4 Cubic Search Method

This method is similar to the successive quadratic point-estimation method discussed in the Section 2.4.1, except that the derivatives are used to reduce the number of required initial points. For example, a cubic function

$$\bar{f}(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2) + a_3(x - x_1)^2(x - x_2)$$

has four unknowns  $a_0$ ,  $a_1$ ,  $a_2$ , and  $a_3$  and, therefore, requires at least four points to determine the function. But the function can also be determined

exactly by specifying the function value as well as the first derivative at only two points:  $((x_1, f_1, f'_1), (x_2, f_2, f'_2))$ . Thereafter, by setting the derivative of the above equation to zero, the minimum of the above function can be obtained (Reklaitis, et al., 1983):

$$\bar{x} = \begin{cases} x_2, & \text{if } \mu = 0, \\ x_2 - \mu(x_2 - x_1), & \text{if } 0 \leq \mu \leq 1, \\ x_1, & \text{if } \mu > 1. \end{cases} \quad (2.11)$$

where

$$\begin{aligned} z &= \frac{3(f_1 - f_2)}{x_2 - x_1} + f'_1 + f'_2, \\ w &= \frac{x_2 - x_1}{|x_2 - x_1|} \sqrt{(z^2 - f'_1 f'_2)}, \\ \mu &= \frac{f'_2 + w - z}{f'_2 - f'_1 + 2w}. \end{aligned}$$

Similar to Powell's successive quadratic estimation method, the minimum of the approximation function  $\bar{f}(x)$  can be used as an estimate of the true minimum of the objective function. This estimate and the earlier two points ( $x_1$  and  $x_2$ ) may be used to find the next estimate of the true minimum point. Two points ( $x_1$  and  $x_2$ ) are so chosen that the product of their first derivative is negative. This procedure may be continued until the desired accuracy is obtained.

### Algorithm

**Step 1** Choose an initial point  $x^{(0)}$ , a step size  $\Delta$ , and two termination parameters  $\epsilon_1$  and  $\epsilon_2$ . Compute  $f'(x^{(0)})$ . If  $f'(x^{(0)}) > 0$ , set  $\Delta = -\Delta$ . Set  $k = 0$ .

**Step 2** Compute  $x^{(k+1)} = x^{(k)} + 2^k \Delta$ .

**Step 3** Evaluate  $f'(x^{(k+1)})$ .

If  $f'(x^{(k+1)})f'(x^{(k)}) \leq 0$ , set  $x_1 = x^{(k)}$ ,  $x_2 = x^{(k+1)}$ , and go to Step 4;

Else set  $k = k + 1$  and go to Step 2.

**Step 4** Calculate the point  $\bar{x}$  using Equation (2.11).

**Step 5** If  $f(\bar{x}) < f(x_1)$ , go to Step 6;

Else set  $\bar{x} = \bar{x} - \frac{1}{2}(\bar{x} - x_1)$  until  $f(\bar{x}) \leq f(x_1)$  is achieved.

**Step 6** Compute  $f'(\bar{x})$ . If  $|f'(\bar{x})| \leq \epsilon_1$  and  $|(\bar{x} - x_1)/\bar{x}| \leq \epsilon_2$ , **Terminate**;

Else if  $f'(\bar{x})f'(x_1) < 0$ , set  $x_2 = \bar{x}$ ;

Else set  $x_1 = \bar{x}$ .

Go to Step 4.

This method is most effective if the exact derivative information is available. The bracketing of the minimum point is achieved in the first three steps. The bracketing algorithm adopted in this method is similar to that of the bounding phase method. Except at the first iteration, in all other iterations the function value as well as the first derivative are calculated only at one new point. Thus, at every iteration, only two new function evaluations are required. The first iteration requires repetitive execution of Steps 2 and 3 to obtain two bracketing points. If the new point  $\bar{x}$  is better than the point  $x_1$ , one of the two points  $x_1$  or  $x_2$  is eliminated depending on which one of them brackets the true minimum with  $\bar{x}$ . If the new point  $\bar{x}$  is worse than the point  $x_1$ , we may find the best two points among  $x_1$ ,  $x_2$ , and  $\bar{x}$  that bracket the minimum point and continue with the algorithm, but, here, excessive derivative computations are avoided by simply modifying the point  $\bar{x}$ .

We take a numerical problem to illustrate the working of the above algorithm.

#### EXERCISE 2.5.4

Consider the following function again:

$$f(x) = x^2 + 54/x.$$

**Step 1** We choose an initial point  $x^{(0)} = 1$ , a step size  $\Delta = 0.5$  and termination parameters  $\epsilon_1 = \epsilon_2 = 10^{-3}$ . The derivative of the function at  $x^{(0)}$  computed numerically is equal to  $f'(x^{(0)}) = -52.005$ . Since  $f'(x^{(0)}) < 0$ , we set  $\Delta = 0.5$ . We also set  $k = 0$ . At this step, we compute the next point  $x^{(1)} = x^{(0)} + 2^0\Delta = 1 + 1(0.5) = 1.5$ . The first point is  $x^{(1)} = 1 + 2^0(0.5) = 1.5$  and  $f'(x^{(1)}) = -21.002$ . Thus, the product  $f'(x^{(0)})f'(x^{(1)}) \leq 0$ .

**Step 2** Thus, we set  $k = 1$  and proceed to Step 2. The second point is  $x^{(2)} = 1.5 + 2^1(0.5) = 2.5$ .

**Step 3** The derivative at this point is  $f'(x^{(2)}) = -3.641$ , which does not make the product negative. We consider the third point  $x^{(3)} = 2.5 + 2^2(0.5) = 4.5$  with derivative  $f'(x^{(3)}) = 6.333$ , which makes the product  $f'(x^{(2)})f'(x^{(3)}) < 0$ . Thus, we set  $x_1 = x^{(2)} = 2.5$  and  $x_2 = x^{(3)} = 4.5$ .

**Step 4** We calculate the estimated point using Equation (2.11). The intermediate parameters are found to be  $z = -3.907$ ,  $w = 6.190$ , and  $\mu = 0.735$ . The estimated point is  $\bar{x} = 4.5 - 0.735(4.5 - 2.5)$  or  $\bar{x} = 3.030$ .

**Step 5** The function value at this point is  $f(\bar{x}) = 27.003$ . Since  $f(\bar{x}) = 27.003 < f(x_1) = 27.850$ , we move to Step 6.

**Step 6** The derivative at the new point is  $f'(\bar{x}) = 0.178$ . Termination criteria are not satisfied at this point. Therefore, we check the products of derivatives among  $\bar{x}$ ,  $x_1$ , and  $x_2$ . It turns out that  $f'(\bar{x})f'(x_1) < 0$ . Hence, we set  $x_2 = \bar{x} = 3.030$ . This completes one iteration of the cubic search algorithm. In order to proceed further, we go to Step 4.

**Step 4** With  $x_1 = 2.5$ ,  $f'(x_1) = -3.641$ ,  $x_2 = 3.030$ , and  $f'(x_2) = 0.178$ , we obtain the estimated point by using Equation (2.11):  $\bar{x} = 2.999$ .

**Step 5** The function value at this point is  $f(\bar{x}) = 27.000$ , which is smaller than  $f(x_1)$ .

**Step 6** The derivative at this point is  $f'(\bar{x}) = -0.007$ . Considering the sign of the derivatives, we set  $x_2 = 3.030$  and  $x_1 = 2.999$  in the next iteration.

Since derivatives are employed, this method is usually faster than Powell's quadratic estimation method, but if derivatives are to be evaluated numerically, Powell's method is preferred.

## 2.6 Root-finding Using Optimization Techniques

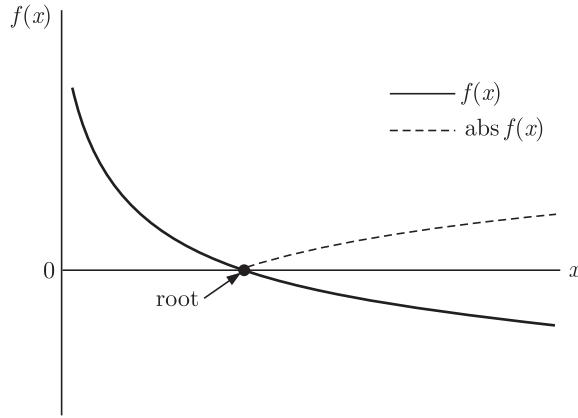
Finding the roots of an equation is common to many engineering activities. In constrained optimization algorithms described in Chapter 4, we shall also come across many root-finding problems that are parts of the optimization process. The root-finding problem can be solved using an optimization technique by suitably choosing an objective function. Consider the following equation in one variable  $x$ :

$$h(x) = 0. \quad (2.12)$$

The root of the above equation is a solution  $x^*$  which will satisfy the above equation, that is, the value of the left expression at the point  $x^*$  is  $h(x^*) = 0$ . For any other solution, the value of that expression will not be zero. When the left expression is zero at a point  $x^*$ , there could be two scenarios. In the close vicinity and either side of the point  $x^*$ , the value of the expression  $h(x)$  is opposite in sign. That is, either the expression  $h(x)$  becomes zero from a positive value and then becomes negative (the solid line in Figure 2.14) or the expression becomes zero from a negative value and then becomes positive. The other scenario is that the expression  $h(x)$  is of the same sign in the close vicinity and either side of the point  $x^*$ . In this case, the point  $x^*$  is an optimum point. The solution procedure in the latter case is the main focus of this book, although the problem in the former scenario can be solved using an optimization technique. We convert the above equation into the following objective function:

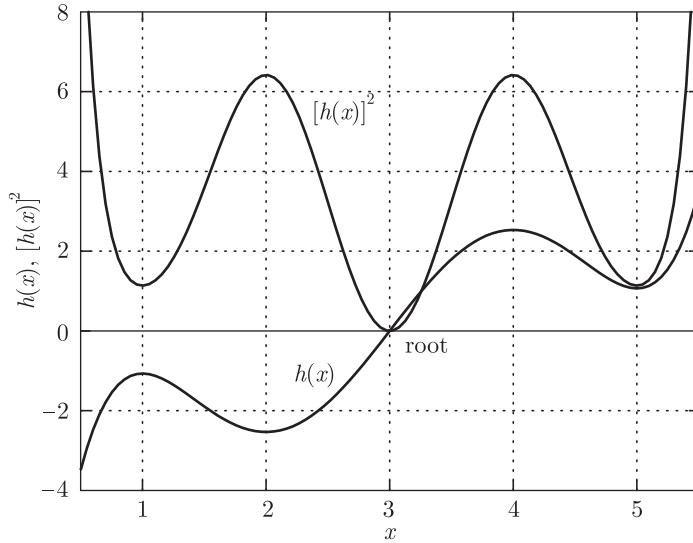
$$\text{Minimize } \text{abs}[h(x)]. \quad (2.13)$$

The values of the above objective function in the vicinity and either side of the point  $x^*$  are now of the same sign (in fact, they are now all positive numbers). In Figure 2.14, the dashed line is the transformed function. Note that other objective functions (like  $[h(x)]^2$ ) would also have achieved the same purpose. The transformation must be such that the root of the original equation becomes the minimum point of the converted optimization problem. The use of  $[h(x)]^2$  has an advantage. The function is differentiable if the original function  $h(x)$  is differentiable; thus, a derivative-based optimization method can be used to find the root of the function. Moreover, in addition to



**Figure 2.14** Transformation of a root-finding problem into an optimization problem.

the stationary points of the original function  $h(x)$ , roots of  $h(x)$  also become minima. Consider the function  $h(x)$  shown in Figure 2.15, which has two minima and two maxima. When the function is squared, these minima and



**Figure 2.15** Transformation of a root-finding problem using a square function.

maxima remain as they are; in addition, there is an additional minimum at the root of the function ( $\bar{x} = 3$ ). Optimality conditions on  $H(x) = [h(x)]^2$  yields the following conditions:

$$\frac{dH}{dx} = h(x)h'(x) = 0, \quad \frac{d^2H}{dx^2} = [h'(x)]^2 + h(x)h''(x).$$

The first condition reveals that either  $h(x) = 0$  or  $h'(x) = 0$ . The former ensures that the root of  $h(x)$  is a stationary point of  $H(x)$  and the latter condition ensures that all stationary points of  $h(x)$  remain stationary. The second-order derivative is positive for  $h(x) = 0$  points, thereby indicating that roots of  $h(x)$  are minima of  $H(x)$  (like the point  $x = 3$  in the figure). However, for an original minimum or maximum point  $\hat{x}$  of  $h(x)$ , there can be two situations. If the  $h(\hat{x})$  is positive (like  $x = 5$  and  $x = 4$  in the example problem), it remains as minimum or maximum of  $H(x)$ , respectively. On the other hand, if  $h(\hat{x})$  is negative, then the minimum becomes maximum and vice versa, as shown for points  $x = 1$  and  $x = 2$ .

Once the optimization problem is formed, a bracketing technique can be used to first bracket the root and then a region-elimination method or a gradient-based search method can be used to find the root with the desired accuracy. We illustrate the root-finding procedure on a simple problem.

### EXERCISE 2.6.1

Let us find the cube-root of a number (say 10). This can be formulated as a root-finding problem as follows:

$$h(x) = x^3 - 10 = 0.$$

A point that satisfies the above equation is the cube-root of number 10. Incidentally, the cube-root of 10 obtained using a calculator is 2.154 (up to three decimal places of accuracy). Let us investigate whether we can get this solution using the above root-finding procedure. We use the bounding phase and golden section search routines given at the end of this chapter to solve this problem. We modify the objective function expression in the subroutine `funct` to code the function  $f(x) = \text{abs}(x^3 - 10)$ . First, the bounding phase method is applied with an initial point  $x^{(0)} = 0$  and  $\Delta = 1$ . (These numbers are chosen at random.) After five function evaluations, the lower and upper limits of the root are found to be 0 and 3, respectively. The input and output of the bounding phase code are shown below:

```
enter x0, delta
0 1
enter 1 for intermediate results
0

The bounds are (      .000,      3.000)
Total function evaluations:      5
```

We now use the golden section search to find the root with three decimal places of accuracy. Using  $\epsilon = 10^{-3}$ ,  $a = 0$ , and  $b = 3$ , we obtain the solution  $x^* = 2.154$ . The input and output of the golden section search code are shown below:

```
Enter lower and upper limits
```

```

0 3
Enter accuracy desired
0.001
Enter 1 for intermediate results
0

Final interval is ( 2.15392E+00, 2.15476E+00)
Accuracy: 2.80034E-04
The estimated minimum is      .21543E+01
Function evaluations required is    18

```

## 2.7 Summary

In this chapter, a number of optimization techniques suitable for finding the minimum point of single-variable functions have been discussed. The problem of finding the minimum point can be divided into two phases. At first, the minimum point of the function needs to be bracketed between a lower and an upper bound. Secondly, the minimum needs to be found as accurately as possible by keeping the search effort enclosed in the bounds obtained in the first phase.

Two different techniques to bracket the minimum point have been discussed. The exhaustive search method requires, in general, more function evaluations to bracket the minimum but the user has a control over the final bracketing range. On the other hand, the bounding phase method can bracket the minimum very fast (usually exponentially fast) but the final bracketing range may be poor.

Once the minimum is bracketed, region-elimination methods, point estimation methods, or gradient-based methods may be used to find a close estimate of the minimum. Region-elimination methods exclude some portion of the search space at every iteration by comparing function values at two points. Among the region-elimination methods discussed in this chapter, the golden section search is the most economical. Gradient-based methods use derivative information, which may be computed numerically. Point estimation methods work by approximating the objective function by simple unimodal functions iteratively, each time finding a better estimate of the true minimum. Two methods—quadratic approximation search based on function values at three points and cubic interpolation search based on function and gradient information at two points—have also been discussed.

For well-behaved objective functions, the convergence to the optimum point is faster with Powell's method than with the region-elimination method. However, for any arbitrary unimodal objective function, the golden section search method is more reliable than other methods described in this chapter.

## REFERENCES

- Powell, M. J. D. (1964). An efficient method for finding the minimum of a function of several variables without calculating derivatives. *Computer Journal*. **7**, 155–162.
- Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. (1983). *Engineering Optimization—Methods and Applications*. New York: Wiley.
- Scarborough, J. B. (1966). *Numerical Mathematical Analysis*. New Delhi: Oxford & IBH Publishing Co.

## PROBLEMS

**2-1** Identify the optimum points of the following functions. Find the optimum function values.

- (i)  $f(x) = x^3 - 10x - 2x^2 + 10.$
- (ii)  $f(x) = (x - 1)^2 - 0.01x^4.$
- (iii)  $f(x) = 2(x - 2) \exp(x - 2) - (x + 3)^2.$
- (iv)  $f(x) = (x^2 - 10x + 2) \exp(0.1x).$
- (v)  $f(x) = 2x - x^3 - 5x^2 - 2 \exp(0.01x).$
- (vi)  $f(x) = 0.01x^5 - 2x^4 + 500(x - 2)^2.$
- (vii)  $f(x) = \exp(x) - x^3.$

**2-2** In order to obtain a polynomial objective function having a saddle point, what is the minimum order of the function needed?

**2-3** Use three iterations of the golden section search method in order to maximize the function

$$f(x) = 10 + x^3 - 2x - 5 \exp(x)$$

in the interval  $(-5, 5)$ .

**2-4** Bracket the minimum of the following functions using the bounding phase method. In all cases, use an initial point  $x^{(0)} = 0$  and an initial  $\Delta = 1$ . Use the computer program listed at the end of this chapter to verify your results. Thereafter, use the golden section search code to find the minimum point with the three decimal places of accuracy.

- (i)  $f(x) = x^2 - 3x - 20.$
- (ii)  $f(x) = \exp((x - 0.2)^2/2).$
- (iii)  $f(x) = 0.1(x^2 - 3x + 5)^2 + (x - 3)^2.$
- (iv)  $f(x) = 2x^4 - x^3 + 5x^2 - 12x + 1.$
- (v)  $f(x) = \exp(x) - 400x^3 + 10.$
- (vi)  $f(x) = (1 - x)^4 - (2x + 10)^2.$
- (vii)  $f(x) = x^3 - 2x + 10.$

**2-5** What proportion of the original search space is retained after 15 function evaluations using (i) golden section search, (ii) interval halving method and (iii) bisection method (with central difference method of gradient computation)?

**2-6** A point estimation method with a sinusoidal function approximation needs to be developed for minimization. A point  $x_1$  is chosen at random in the search interval so that two other points  $x_2 = x_1 + \Delta$  and  $x_3 = x_1 + 2\Delta$  also remain in the search space. The function values at these points are  $f_1$ ,  $f_2$ , and  $f_3$ , respectively. An approximating function

$$s(x) = a_0 + a_1 \sin a_2(x - x_1)$$

is chosen in the interval  $(\pi/2, 5\pi/2)$ .

- (i) Find the estimate  $(\bar{x})$  of the minimum in terms of the above parameters.
- (ii) What relationships among these parameters are required in order to make the search method *successful*?
- (iii) Use the above approximation function to find an estimate of the minimum of the function  $f(x) = (x - 3)^2$  with  $x_1 = 1.5$  and  $\Delta = 1$ .

**2-7** We would like to use the following approximating second-order polynomial

$$q(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2)$$

to match the objective function values  $f_1$  and  $f_2$  at two points  $x_1$  and  $x_2$  (with  $x_2 > x_1$ ), respectively, and the first derivative  $f'_1$  at  $x_1$ .

- (i) Find  $a_0$ ,  $a_1$ , and  $a_2$  in terms of  $x_1$ ,  $x_2$ ,  $f_1$ ,  $f_2$ , and  $f'_1$ .
- (ii) Find the minimum of the approximating function.
- (iii) In finding the minimum of  $f(x) = x^3 - 3x + 2$ , we would like to use the above approximating polynomial  $q(x)$  for the following two scenarios:
  - (a)  $x_1 = 2$  and  $x_2 = 3$ . Find the approximating minimum solution obtained after one iteration of this approach.
  - (b)  $x_1 = -5$  and  $x_2 = 5$ . Explain why we cannot use these points to successfully find the minimum of  $f(x)$ .

**2-8** Use two iterations of Powell's quadratic estimation method to minimize the following function:

$$f(x) = 2 \exp(x) - x^3 - 10x.$$

**2-9** Use three iterations of the bisection and the secant method to minimize the following function:

$$\exp(0.2x) - (x + 3)^2 - 0.01x^4.$$

Compare the algorithms in terms of the interval obtained at the end of three iterations.

**2-10** Compare the golden section search and interval halving method in terms of the obtained interval after 10 function evaluations for the minimization of the function

$$f(x) = x^2 - 10 \exp(0.1x)$$

in the interval  $(-10, 5)$ .

**2-11** Compare the bisection and secant methods in terms of the obtained interval after 10 function evaluations for the minimization of the function

$$f(x) = \exp(x) - x^3$$

in the interval  $(2, 5)$ . How does the outcome change if the interval  $(-2, 5)$  is chosen?

**2-12** Find at least one root of the following functions:

- (i)  $f(x) = x^3 + 5x^2 - 3$ .
- (ii)  $f(x) = (x + 10)^2 - 0.01x^4$ .
- (iii)  $f(x) = \exp(x) - x^3$ .
- (iv)  $f(x) = (2x - 5)^4 - (x^2 - 1)^3$ .
- (v)  $f(x) = ((x + 2)^2 + 10)^2 - x^4$ .

**2-13** Perform two iterations of the cubic search method to minimize the function

$$f(x) = (x^2 - 1)^3 - (2x - 5)^4.$$

**2-14** In trying to solve the following problem using *feasible direction method*<sup>1</sup> at the current point  $x^{(t)} = (0.5, 0.5)^T$ , a direction vector  $d^{(t)} = (1, 0.25)^T$  is obtained.

$$\text{Minimize } (x_1 - 1)^2 + (x_2 - 3)^2$$

subject to

$$4.5x_1 + x_2^2 - 18 \leq 0,$$

$$2x_1 - x_2 - 1 \geq 0,$$

$$x_1, x_2 \geq 0.$$

Use two iterations of the interval halving method to find the bracketing points. Find out the exact minimum point along  $d^{(t)}$  and compare.

---

<sup>1</sup>The feasible direction method is discussed in Chapter 4.

## COMPUTER PROGRAMS

In this section, we present two FORTRAN codes implementing the bounding phase algorithm and the golden section search algorithm. These codes are written in a step-by-step format as described in Sections 2.2.2 and 2.3.3. These codes show how other algorithms outlined in this chapter can be easily coded. We first present the code for the bounding phase algorithm and then present the code for the golden section search method. Sample runs showing the working of the codes are also presented.

### Bounding phase method

The bounding phase algorithm is coded in subroutine **bphase**. The objective function is coded in function **funct**. In the given code, the objective function  $f(x) = x^2 + 54/x$  is used. The user needs to modify the function **funct** for a different objective function.

```

c      ****
c      *
c      *          BOUNDING PHASE METHOD
c      *
c      ****
c      Developed by Dr. Kalyanmoy Deb
c                      Indian Institute of Technology, Kanpur
c All rights reserved. This listing is for personal use.
c%%%%%%%%%%%%%%%
c      Change the function funct() for a new function
c%%%%%%%%%%%%%%%
      implicit real*8 (a-h,o-z)
      nfun = 0
      call bphase(a,b,nfun)
      write(*,5) a,b,nfun
5   format(2x,'The bounds are (',f10.3,', ',f10.3,')',
      -           /,2x,'Total function evaluations:',i6)
      stop
      end

      subroutine bphase(a,b,nfun)
c      bounding phase algorithm
c      ****
c      a and b are lower and upper bounds (output)
c      nfun is the number of function evaluations (output)
c      ****
      implicit real*8 (a-h,o-z)
c.....step 1 of the algorithm
1   write(*,*) 'enter x0, delta'
      read(*,*) x0,delta
      call funct(x0-delta,fn,nfun)

```

```

call funct(x0,f0,nfun)
call funct(x0+delta,fp,nfun)
write(*,*) 'enter 1 for intermediate results'
read(*,*) iprint
c.....step 2 of the algorithm
if (fn .ge. f0) then
  if (f0 .ge. fp) then
    delta = 1 * delta
  else
    a = x0 - delta
    b = x0 + delta
  endif
elseif ((fn .le. f0) .and. (f0 .le. fp)) then
  delta = -1 * delta
else
  go to 1
endif
k=0
xn = x0 - delta
c.....step 3 of the algorithm
3  x1 = x0 + (2**k) * delta
call funct(x1,f1,nfun)
if (iprint .eq. 1) then
  write(*,4) x1, f1
4  format(2x,'Current point ',f10.4,
         ' function value ',1pe15.4)
endif
c.....step 4 of the algorithm
if (f1 .lt. f0) then
  k = k+1
  xn = x0
  fn = f0
  x0 = x1
  f0 = f1
  go to 3
else
  a = xn
  b = x1
endif
if (b .lt. a) then
  temp = a
  a = b
  b = temp
endif
return
end

```

```

        subroutine funct(x,f,nfun)
c ****
c x is the current point (input)
c f is the function value (output)
c nfun if the number of function evaluations (output)
c ****
      implicit real*8 (a-h,o-z)
      nfun = nfun + 1
      f = x*x + 54.0/x
      return
      end

```

### Simulation run

The above code is run on a PC-386 under Microsoft FORTRAN compiler and the following results are obtained. An initial  $x^{(0)} = 0.6$  and  $\Delta = 0.5$  are specified. Intermediate points created by the algorithm are also shown.

```

enter x0, delta
0.6 0.5
enter 1 for intermediate results
1

Current point      1.1000 function value      5.0301E+01
Current point      2.1000 function value      3.0124E+01
Current point      4.1000 function value      2.9981E+01
Current point      8.1000 function value      7.2277E+01
The bounds are (    2.100,      8.100)
Total function evaluations:      7

```

### Golden section search

The golden section search algorithm is coded in subroutine `golden`. The function  $f(x) = x^2 + 54/x$  is coded in the function `funct` which can be modified for a different objective function.

```

c ****
c *
c *          GOLDEN SECTION SEARCH
c *
c ****
c Developed by Kalyanmoy Deb
c           Indian Institute of Technology, Kanpur
c All rights reserved. This listing is for personal use.
c This routine finds the minimum point of a single

```

```

c variable function in a specified interval using golden
c section search algorithm.
c Code your function in subroutine funct
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      implicit real*8 (a-h,o-z)
      write(*,*) 'Enter lower and upper limits'
      read(*,*) a,b
      write(*,*) 'Enter accuracy desired'
      read(*,*) eps
      write(*,*) 'Enter 1 for intermediate results'
      read(*,*) iprint
c.....call the golden section search routine
      call golden(a,b,eps/(b-a),xstar,nfun,ierr,iprint)
      if (ierr .ne. 1) then
         write(*,1) xstar
1        format(2x,'The estimated minimum is ',e14.5)
         write(*,2) nfun
2        format(2x,'Function evaluations required is',i6)
      else
         write(*,*) 'Some problem in computation'
      endif
      stop
      end

      subroutine golden(a,b,eps,xstar,nfun,ierr,iprint)
c ****
c a and b are lower and upper limits of the search
c interval eps is the final accuracy desired
c xstar is the obtained solution (output)
c nfun is the number of function evaluations (output)
c ****
      implicit real*8 (a-h,o-z)
      real*8 lw
c.....step 1 of the algorithm
      xstar = a
      ierr=0
      maxfun = 10000
      aw=0.0
      bw=1.0
      lw=1.0
      nfun=0
      k=1
      gold=(sqrt(5.0)-1.0)/2.0
      w1prev = gold
      w2prev = 1.0-gold
      call funct(a,b,w1prev,fw1,nfun)

```

```

call funct(a,b,w2prev, fw2, nfun)
ic=0
c.....step 2 of the algorithm
10 w1 = w1prev
    w2 = w2prev
    if (ic .eq. 1) then
        fw2 = fw1
        call funct(a,b,w1, fw1, nfun)
    else if (ic .eq. 2) then
        fw1 = fw2
        call funct(a,b,w2, fw2, nfun)
    else if (ic .eq. 3) then
        call funct(a,b,w1, fw1, nfun)
        call funct(a,b,w2, fw2, nfun)
    endif
    if (fw1 .lt. fw2) then
c..... first scenario
        ic = 1
        aw = w2
        lw = bw-aw
        w1prev = aw + gold * lw
        w2prev = w1
c..... second scenario
    else if (fw2 .lt. fw1) then
        ic = 2
        bw = w1
        lw=bw-aw
        w1prev = w2
        w2prev = bw - gold * lw
c..... third scenario
    else
        ic = 3
        aw = w2
        bw = w1
        lw = bw-aw
        w1prev = aw + gold * lw
        w2prev = bw - gold * lw
    endif
c.....print intermediate solutions
    if (iprint .eq. 1) then
        write(*,9) a+aw*(b-a), a+bw*(b-a)
9     format(2x,'Current interval is (',1pe12.4,', ',',
-                   1pe12.4,')')
    endif
    k=k+1
c.....step 3 of the algorithm

```

```

        if (dabs(lw) .lt. eps) then
          write(*,*) '-----',
          write(*,2) a+aw*(b-a), a+bw*(b-a),lw
2       format(2x,'Final interval is (',1pe12.5,',',
          -           1pe12.5,')',/,2x,'Accuracy: ',1pe12.5)
          xstar = a + (b-a) * (aw+bw)/2
          return
        else if (nfun .gt. maxfun) then
          write(*,3) maxfun
3       format('The algorithm did not converge in',i6,
          -           ' function evaluations')
          write(*,2) a+aw*(b-a), a+bw*(b-a)
          ierr = 1
          return
        endif
        go to 10
      end

      subroutine funct(a,b,w,fw,nfun)
c ****
c a and b are lower and upper limits
c w is the current point
c fw is the function value (output)
c nfun if the current number of function evaluations
c ****
      implicit real*8 (a-h,o-z)
c.....mapping from w to x
      x = a + w * (b-a)
c.....calculate the function value, change here
c      fw = x*x + 54.0/x
      x1 = sqrt(4.84-(x-2.5)**2) + 0.05
      fw = (x1**2 + x - 11.0)**2 + (x1 + x*x - 7)**2
      nfun = nfun + 1
      return
    end

```

### Simulation run

The above code is run on a PC-386 under Microsoft FORTRAN compiler with the bounds obtained from the simulation run on the bounding phase method. Thus, we input  $a = 2.1$  and  $b = 8.1$ . The accuracy is set to  $10^{-3}$  in order to get the solution with three decimal places of accuracy. The input and output statements are shown below.

```

Enter lower and upper limits
0 5

```

```
Enter accuracy desired
0.001
Enter 1 for intermediate results
1

Current interval is ( 1.9098E+00,      5.0000E+00)
Current interval is ( 1.9098E+00,      3.8197E+00)
Current interval is ( 2.6393E+00,      3.8197E+00)
Current interval is ( 2.6393E+00,      3.3688E+00)
Current interval is ( 2.6393E+00,      3.0902E+00)
Current interval is ( 2.8115E+00,      3.0902E+00)
Current interval is ( 2.9180E+00,      3.0902E+00)
Current interval is ( 2.9180E+00,      3.0244E+00)
Current interval is ( 2.9586E+00,      3.0244E+00)
Current interval is ( 2.9837E+00,      3.0244E+00)
Current interval is ( 2.9837E+00,      3.0089E+00)
Current interval is ( 2.9933E+00,      3.0089E+00)
Current interval is ( 2.9933E+00,      3.0029E+00)
Current interval is ( 2.9970E+00,      3.0029E+00)
Current interval is ( 2.9993E+00,      3.0029E+00)
Current interval is ( 2.9993E+00,      3.0015E+00)
Current interval is ( 2.9993E+00,      3.0007E+00)
Current interval is ( 2.9998E+00,      3.0007E+00)
Final interval is ( 2.99980E+00, 3.00067E+00)
Accuracy: 1.73070E-04
The estimated minimum is      .30002E+01
Function evaluations required is     19
```

# 3

## Multivariable Optimization Algorithms

---

In this chapter, we present algorithms for optimizing functions having multiple design or decision variables. Single-variable function optimization algorithms described in Chapter 2 are used in some of these algorithms to perform a unidirectional search along a desired direction. Therefore, a proper understanding of the single-variable function optimization algorithms would be helpful in appreciating multivariable function optimization algorithms presented in this chapter. The algorithms are presented for minimization problems, but they can also be used for maximization problems by using the duality principle described in Chapter 1.

The algorithms can be broadly classified into two categories—direct search methods and gradient-based methods. In direct search methods, only function values at different points are used to constitute a search. In gradient-based methods, derivative information is used to constitute a search. Before we present the algorithms, let us discuss the optimality criteria for a point in multivariable functions.

### 3.1 Optimality Criteria

The definition of a local, a global, or an inflection point remains the same as that for single-variable functions, but the optimality criteria for multivariable functions are different. In a multivariable function, the gradient of a function is not a scalar quantity; instead it is a vector quantity. The optimality criteria can be derived by using the definition of a local optimal point and by using Taylor's series expansion of a multivariable function (Rao, 1984). Without going into the details of the analysis, we simply present the optimality criteria for a multivariable function.

In this chapter and subsequent chapters, we assume that the objective function is a function of  $N$  variables represented by  $x_1, x_2, \dots, x_N$ . The gradient vector at any point  $x^{(t)}$  is represented by  $\nabla f(x^{(t)})$  which is an  $N$ -dimensional vector given as follows:

$$\nabla f(x^{(t)}) = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_N} \right)^T \Big|_{x^{(t)}}. \quad (3.1)$$

The first-order partial derivatives can be calculated numerically using Equation (2.5) presented in the previous chapter. However, the second-order derivatives in multivariable functions form a matrix,  $\nabla^2 f(x^{(t)})$  (better known as the Hessian matrix) given as follows:

$$\nabla^2 f(x^{(t)}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix} \Big|_{x^{(t)}}. \quad (3.2)$$

The second-order partial derivatives can also be calculated numerically. We shall discuss this in detail in Section 3.4. By defining the derivatives, we are now ready to present the optimality criteria (Reklaitis et al., 1983):

*A point  $\bar{x}$  is a stationary point if  $\nabla f(\bar{x}) = 0$ . Furthermore, the point is a minimum, a maximum, or an inflection point if  $\nabla^2 f(\bar{x})$  is positive-definite, negative-definite, or otherwise.*

A matrix  $\nabla^2 f(x^{(t)})$  is defined to be positive-definite if for any point  $y$  in the search space the quantity  $y^T \nabla^2 f(x^{(t)}) y \geq 0$ . The matrix is called a negative definite matrix if at any point the quantity  $y^T \nabla^2 f(x^{(t)}) y \leq 0$ . If at some point  $y^+$  in the search space the quantity  $y^{+T} \nabla^2 f(x^{(t)}) y^+$  is positive and at some other point  $y^-$  the quantity  $y^{-T} \nabla^2 f(x^{(t)}) y^-$  is negative, then the matrix  $\nabla^2 f(x^{(t)})$  is neither positive-definite nor negative-definite. There are a number of ways to investigate whether a matrix is positive-definite (Strang, 1980). One common way to do this is to evaluate the eigenvalues of the matrix. If all eigenvalues are positive, the matrix is positive-definite. The other way to test the positive-definiteness of a matrix is to calculate the principal determinants of the matrix. If all principal determinants are positive, the matrix is positive-definite. It is worth mentioning here that the negative-definiteness of a matrix  $A$  can be verified by testing the positive-definiteness of the matrix  $-A$ . Further discussion on this aspect is beyond the scope of this book. Interested readers may refer to matrix algebra texts for more information (Kreyszig, 1983; Strang, 1980).

### 3.2 Unidirectional Search

Many multivariable optimization techniques use successive unidirectional search techniques to find the minimum point along a particular search direction. Since unidirectional searches will be mentioned and used a number of times in the remaining chapters, we illustrate here how a unidirectional search can be performed on a multivariable function.

A unidirectional search is a one-dimensional search performed by comparing function values only along a specified direction. Usually, a unidirectional search is performed from a point  $x^{(t)}$  and in a specified direction  $s^{(t)}$ . That is, only points that lie on a line (in an  $N$ -dimensional space) passing through the point  $x^{(t)}$  and oriented along the search direction  $s^{(t)}$  are allowed to be considered in the search process. Any arbitrary point on that line can be expressed as follows:

$$x(\alpha) = x^{(t)} + \alpha s^{(t)}. \quad (3.3)$$

The parameter  $\alpha$  is a scalar quantity, specifying a relative measure of distance of the point  $x(\alpha)$  from  $x^{(t)}$ . Note, however, that the above equation is a vector equation specifying all design variables  $x_i(\alpha)$ . Thus, for a given value of  $\alpha$ , the point  $x(\alpha)$  can be known. Any positive and negative value of  $\alpha$  will create a point on the desired line. If  $\alpha = 0$ , the current point  $x^{(t)}$  is obtained. In order to find the minimum point on the specified line, we can rewrite the multivariable objective function in terms of a single variable  $\alpha$  by substituting each variable  $x_i$  by the expression  $x_i(\alpha)$  given in Equation (3.3) and by using a suitable single-variable search method described in Chapter 2. Once the optimal value  $\alpha^*$  is found, the corresponding point can also be found using Equation (3.3). Let us illustrate the working of the unidirectional search technique on a two-variable function optimization problem.

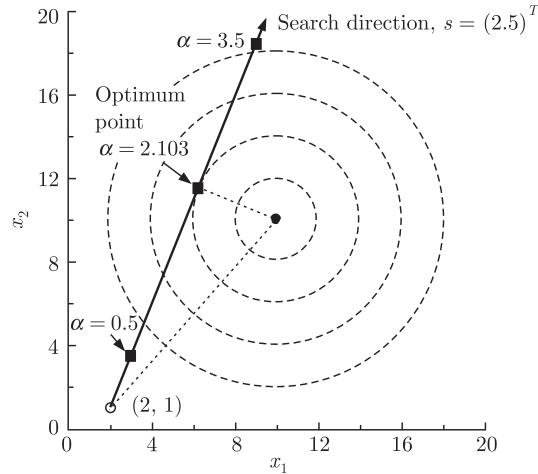
#### EXERCISE 3.2.1

Consider the objective function:

$$\text{Minimize } f(x_1, x_2) = (x_1 - 10)^2 + (x_2 - 10)^2.$$

Figure 3.1 shows the contour plot of this function. The property of a contour line is that any two points on a contour line have the same function value. Thus, it is convenient to show the optimum point of a function on contour plots. The figure shows that the minimum point lies at the point  $(10, 10)^T$ . The function value at this point is zero. Let us say that the current point of interest is  $x^{(t)} = (2, 1)^T$  and we are interested in finding the minimum point and the minimum function value in a search direction  $s^{(t)} = (2, 5)^T$  from the current point. From the right-angled triangle shown in dotted line, we obtain the optimal point  $x^* = (6.207, 11.517)^T$ . Let us investigate whether we can find this solution by performing a unidirectional search along  $s^{(t)}$  on the function.

The search can be achieved by first using a bracketing algorithm to enclose the optimum point and then using a single-variable function optimization



**Figure 3.1** An illustration of a unidirectional search method.

technique to find the optimum point with the desired accuracy. Here, we use the bounding phase method and the golden section search method for these purposes, respectively. The code for the bounding phase method presented at the end of Chapter 2 is used with the following modifications to the subroutine `funct`:

```

subroutine funct(x,f,nfun)
implicit real*8 (a-h,o-z)
dimension xt(2),s(2),xl(2)
c      xt : current point (a vector)
c      s : specified direction vector
c      xl : a point along the search direction
      data xt/2.0,1.0/
      data s/2.0,5.0/
      nfun = nfun + 1
      do 10 i = 1,2
         xl(i) = xt(i) + x * s(i)
10   continue
      f = (xl(1)-10)**2 + (xl(2)-10)**2
      return
      end

```

The input to the code and the corresponding results obtained by running the code are shown as follows:

```

enter x0, delta
0 0.5
enter 1 for intermediate results
0

```

```
The bounds are (      .500,      3.500)
Total function evaluations:      6
```

The bounding phase algorithm finds the bracketing points  $(3.0, 3.5)^T$  and  $(9.0, 18.5)^T$ , corresponding to  $\alpha = 0.5$  and  $\alpha = 3.5$ , respectively. Note that both these points lie on the specified line (Figure 3.1). We now use the golden section search to find the optimum point with three decimal places of accuracy. The function `funct` in the code for golden section search method presented at the end of Chapter 2 is also replaced by the above code. Using  $a = 0.5$ , and  $b = 3.5$  (for  $\alpha$ ), we obtain the optimum solution with three decimal places of accuracy:  $\alpha^* = 2.103$ . The input and output of the code are shown below:

```
Enter lower and upper limits
0.5 3.5
Enter accuracy desired
0.001
Enter 1 for intermediate results
0

Final interval is ( 2.10307E+00, 2.10391E+00)
Accuracy: 2.80034E-04
The estimated minimum is     .21035E+01
Function evaluations required is    18
```

Substituting  $\alpha^* = 2.103$ ,  $x^{(t)} = (3.0, 3.5)^T$  and  $s^{(t)} = (2, 5)^T$  in Equation (3.3), we obtain  $x^* = (6.207, 11.517)^T$ , which is the same as that found using geometric properties. At the end of this chapter, we present a multivariable function optimization code, where the above unidirectional search procedure is coded in a different (and better) way.

### 3.3 Direct Search Methods

In this section, we present a number of minimization algorithms that use function values only. Algorithms requiring gradient of the objective function are discussed in the next section. It is important to mention here that if the gradient information is available, a gradient-based method may be more efficient. Unfortunately, many real-world optimization problems require the use of computationally expensive simulation packages to calculate the objective function, thereby making it difficult to compute the derivative of the objective function. In these problems, direct search techniques may be found to be useful.

In a single-variable function optimization, there are only two search directions a point can be modified—either in the positive  $x$ -direction or the negative  $x$ -direction. The extent of increment or decrement in each direction depends on the current point and the objective function. In multi-objective function optimization, each variable can be modified either in the positive or in the negative direction, thereby totaling  $2^N$  different ways. Moreover,

an algorithm, having searches along each variable one at a time, can only successfully solve linearly separable functions. These algorithms (called one-variable-at-a-time methods) cannot usually solve functions having nonlinear interactions among design variables. Ideally, we require algorithms which either completely eliminate the concept of search direction and manipulate a set of points to create a better set of points or use complex search directions to effectively decouple the nonlinearity of the function. In the following subsections, we describe two algorithms of each kind.

### 3.3.1 Box's Evolutionary Optimization Method

Box's evolutionary optimization is a simple optimization technique developed by G. E. P. Box<sup>1</sup> in 1957. The algorithm requires  $(2^N + 1)$  points, of which  $2^N$  are corner points of an  $N$ -dimensional hypercube<sup>2</sup> centred on the other point. All  $(2^N + 1)$  function values are compared and the best point is identified. In the next iteration, another hypercube is formed around this best point. If at any iteration, an improved point is not found, the size of the hypercube is reduced. This process continues until the hypercube becomes very small.

#### Algorithm

**Step 1** Choose an initial point  $x^{(0)}$  and size reduction parameters  $\Delta_i$  for all design variables,  $i = 1, 2, \dots, N$ . Choose a termination parameter  $\epsilon$ . Set  $\bar{x} = x^{(0)}$ .

**Step 2** If  $\|\Delta\| < \epsilon$ , Terminate;

Else create  $2^N$  points by adding and subtracting  $\Delta_i/2$  from each variable at the point  $\bar{x}$ .

**Step 3** Compute function values at all  $(2^N + 1)$  points. Find the point having the minimum function value. Designate the minimum point to be  $\bar{x}$ .

**Step 4** If  $\bar{x} = x^{(0)}$ , reduce size parameters  $\Delta_i = \Delta_i/2$  and go to Step 2;

Else set  $x^{(0)} = \bar{x}$  and go to Step 2.

In the above algorithm,  $x^{(0)}$  is always set as the current best point. Thus, at the end of simulation,  $x^{(0)}$  becomes the obtained optimum point. It is evident from the algorithm that at most  $2^N$  functions are evaluated at each iteration. Thus, the required number of function evaluations increases exponentially with  $N$ . The algorithm, however, is simple to implement and has had success in solving many industrial optimization problems (Box, 1957; Box and Draper, 1969). We illustrate the working of this algorithm through an exercise problem.

---

<sup>1</sup>This algorithm should not be confused with well-established evolutionary optimization field (Goldberg, 1989; Holland, 1975). We discuss one evolutionary optimization method, largely known as Genetic Algorithms (GAs), in Chapter 6.

<sup>2</sup>An  $N$ -dimensional hypercube is an  $N$ -dimensional box, whose length in each dimension is fixed according to the precision required in the respective variable.

**EXERCISE 3.3.1**

Consider the Himmelblau function (Reklaitis et al., 1983):

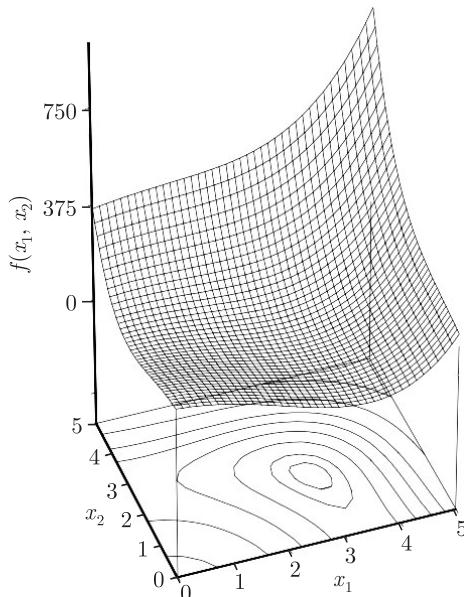
$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

in the interval  $0 \leq x_1, x_2 \leq 5$ . This function is chosen throughout this text for a particular reason. The function is a summation of two squared terms. Each term inside the bracket can be considered as an error term. The first term calculates the difference between the term  $(x_1^2 + x_2)$  and 11 and the second term calculates the difference between the term  $(x_1 + x_2^2)$  and 7. Since the objective is to minimize these squared differences (or deviations of the two terms from 11 and 7, respectively), the optimum solution will be a set of values of  $x_1$  and  $x_2$ , satisfying the following two equations:

$$x_1^2 + x_2 = 11, \quad x_1 + x_2^2 = 7.$$

Many engineering design problems aim to find a set of design parameters satisfying a number of goals simultaneously. In these problems, a mathematical expression for each goal is usually written and the difference of the expression from the target is calculated. The differences are then squared and added together to form an overall objective function, which must be minimized. Thus, the above Himmelblau function resembles the mathematical expression of an objective function in many engineering design problems.

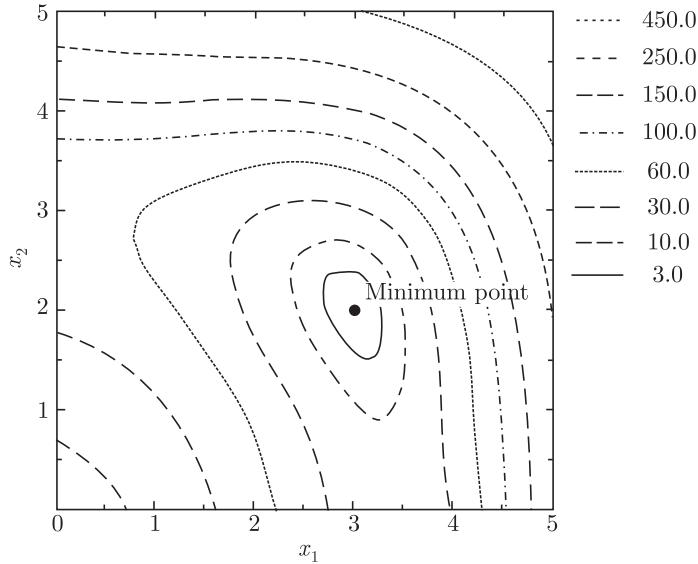
Himmelblau's function is plotted in Figure 3.2 in the range  $0 \leq x_1, x_2 \leq 5$ . It appears from the plot that the minimum point is at  $(3, 2)^T$ . The minimum



**Figure 3.2** A plot of the Himmelblau function vs.  $x_1$  and  $x_2$ . The contour plot of the function shows that the minimum point is at  $(3, 2)^T$ .

point can also be obtained by solving the above two equations. The function value at this minimum point is equal to zero. The contour of the function is also shown at the bottom of the function plot.

A contour line is a collection of points having identical function values. The same contour plot is shown in Figure 3.3. The continual decrease in the function value of successive contour lines as they approach a point means that the point is the minimum point. Thus, a contour plot gives an efficient (visual) means to identify the minimum points in an objective function. Throughout this text, we shall demonstrate the progress of optimization algorithms on the contour plot shown in Figure 3.3.



**Figure 3.3** The contour plot of the Himmelblau function. The function value corresponding to each contour line is also listed.

**Step 1** We choose an initial point  $x^{(0)} = (1, 1)^T$  and a size reduction parameter,  $\Delta = (2, 2)^T$ . We also choose  $\epsilon = 10^{-3}$  and initialize  $\bar{x} = x^{(0)} = (1, 1)^T$ .

**Step 2** Since  $\|\Delta\| = 2.828 > 10^{-3}$ , we create a two-dimensional hypercube (a square) around  $x^{(0)}$ :

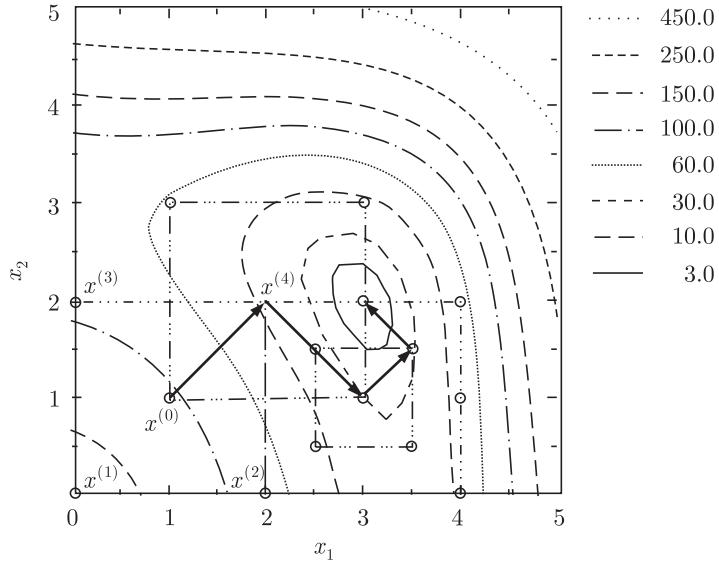
$$x^{(1)} = (0, 0)^T, \quad x^{(2)} = (2, 0)^T, \quad x^{(3)} = (0, 2)^T, \quad x^{(4)} = (2, 2)^T.$$

Figure 3.4 shows this square with five chosen points.

**Step 3** The function values at the five points are

$$\begin{aligned} f(x^{(0)}) &= 106, & f(x^{(1)}) &= 170, & f(x^{(2)}) &= 74, \\ f(x^{(3)}) &= 90, & f(x^{(4)}) &= 26. \end{aligned}$$

The minimum of the five function values is 26 and the corresponding point is  $x^{(4)} = (2, 2)^T$ . Thus we designate  $\bar{x} = (2, 2)^T$ .



**Figure 3.4** Five iterations of the Box's evolutionary optimization method shown on a contour plot of Himmelblau's function.

**Step 4** Since  $\bar{x} \neq x^{(0)}$ , we set  $x^{(0)} = (2, 2)^T$  and go to Step 2. This completes one iteration of the Box's evolutionary optimization method. Figure 3.4 shows how the initial point  $(1, 1)^T$  has moved to the point  $(2, 2)^T$  in one iteration.

**Step 2** The quantity  $\|\Delta\| = 2.828$  is not small and therefore, we create a square around the point  $(2, 2)^T$  by adding and subtracting  $\Delta_i$  to the variable  $x_i$ :

$$x^{(1)} = (1, 1)^T, \quad x^{(2)} = (3, 1)^T, \quad x^{(3)} = (1, 3)^T, \quad x^{(4)} = (3, 3)^T.$$

**Step 3** The corresponding function values are

$$f(x^{(0)}) = 26, \quad f(x^{(1)}) = 106, \quad f(x^{(2)}) = 10,$$

$$f(x^{(3)}) = 58, \quad f(x^{(4)}) = 26.$$

The minimum point is  $\bar{x} = (3, 1)^T$  having the function value equal to 10.

**Step 4** Since  $\bar{x} \neq x^{(0)}$ , we set the current best point  $x^{(0)} = (3, 1)^T$  and proceed to Step 2.

**Step 2** The new hypercube around  $(3, 1)^T$  is as follows:

$$x^{(1)} = (2, 0)^T, \quad x^{(2)} = (4, 0)^T, \quad x^{(3)} = (2, 2)^T, \quad x^{(4)} = (4, 2)^T.$$

**Step 3** The corresponding function values are  $f(x^{(0)}) = 10$ ,  $f(x^{(1)}) = 74$ ,  $f(x^{(2)}) = 34$ ,  $f(x^{(3)}) = 26$ , and  $f(x^{(4)}) = 50$ . The minimum point is  $\bar{x} = (3, 1)^T$  having the function value equal to 10.

**Step 4** Since the new point is the same as the previous best point  $x^{(0)}$ , we reduce the size parameter  $\Delta = (1, 1)^T$  and move to Step 2.

**Step 2** The new square with a reduced size around  $(3, 1)^T$  is as follows (Figure 3.4):

$$\begin{aligned}x^{(1)} &= (2.5, 0.5)^T, & x^{(2)} &= (3.5, 0.5)^T, \\x^{(3)} &= (2.5, 1.5)^T, & x^{(4)} &= (3.5, 1.5)^T.\end{aligned}$$

**Step 3** The minimum of these points is  $\bar{x} = x^{(4)} = (3.5, 1.5)^T$  having a function value of 9.125.

**Step 4** By reducing the size of the rectangle, we get a better point compared to the previous best point. We proceed to Step 2 by setting  $x^{(0)} = (3.5, 1.5)^T$ .

**Step 2** The new square around  $(3.5, 1.5)^T$  is as follows:

$$x^{(1)} = (3, 1)^T, \quad x^{(2)} = (4, 1)^T, \quad x^{(3)} = (3, 2)^T, \quad x^{(4)} = (4, 2)^T.$$

**Step 3** The minimum point is  $\bar{x} = (3, 2)^T$  having a function value equal to zero.

**Step 4** The point  $\bar{x}$  is better than the previous best point and we proceed to Step 2 by setting  $x^{(0)} = (3, 2)^T$ . Figure 3.4 shows the progress of the algorithm by indicating the movement of the best point from one iteration to another using arrows.

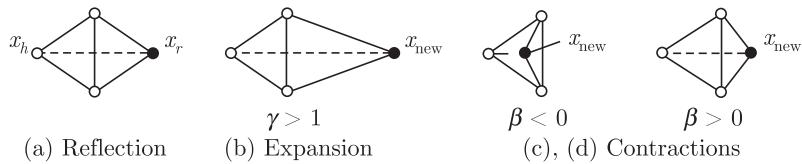
It is interesting to note that although the minimum point is found, the algorithm does not terminate at this step. Since the current point is the minimum, no other point can be found better than  $x^{(0)} = (3, 2)^T$  and therefore, in subsequent iterations the value of the size parameter will continue to decrease (according to Step 4 of the algorithm). When the value  $\|\Delta\|$  becomes smaller than  $\epsilon$ , the algorithm terminates.

It is clear from the working of the algorithm that its convergence depends on the initial hypercube size and location, and the chosen size reduction parameter  $\Delta_i$ . Starting with a large  $\Delta_i$  is good, but the convergence to the minimum may require more iterations and hence more function evaluations. On the other hand, starting with a small hypercube may lead to premature convergence on a suboptimal point, especially in the case of highly nonlinear functions. Even with a large initial hypercube, the algorithm does not guarantee convergence to a local or global optimal solution. It is worth noting here that the reduction of the size parameter ( $\Delta_i$ ) by a factor of two in Step 4 is not always necessary. A smaller or larger reduction can be used. However, a smaller reduction (a factor smaller than two and greater than one) is usually recommended for a better convergence.

### 3.3.2 Simplex Search Method

In the simplex search method, the number of points in the initial *simplex* is much less compared to that in Box's evolutionary optimization method. This reduces the number of function evaluations required in each iteration. With  $N$  variables only  $(N + 1)$  points are used in the initial simplex. Even though some guidelines are suggested to choose the initial simplex (Reklaitis et al., 1983), it should be kept in mind that the points chosen for the initial simplex should not form a zero-volume  $N$ -dimensional hypercube. Thus, in a function with two variables, the chosen three points in the simplex should not lie along a line. Similarly, in a function with three variables, four points in the initial simplex should not lie on a plane.

At each iteration, the worst point in the simplex is found first. Then, a new simplex is formed from the old simplex by some fixed rules that steer the search away from the worst point in the simplex. The extent of steering depends on the relative function values of the simplex. Four different situations may arise depending on the function values. The situations are depicted in Figure 3.5.



**Figure 3.5** An illustration of the simplex search method. First a reflection is performed (a). Depending on function values, an expansion (b) and two different contractions [(c) and (d)] are possible.

At first, the centroid ( $x_c$ ) of all but the worst point is determined. Thereafter, the worst point in the simplex is reflected about the centroid and a new point  $x_r$  is found. The reflection operation is depicted in Figure 3.5(a). If the function value at this point is better than the best point in the simplex, the reflection is considered to have taken the simplex to a good region in the search space. Thus, an *expansion* along the direction from the centroid to the reflected point is performed (Figure 3.5(b)). The amount of expansion is controlled by the factor  $\gamma$ . On the other hand, if the function value at the reflected point is worse than the worst point in the simplex, the reflection is considered to have taken the simplex to a bad region in the search space. Thus, a *contraction* in the direction from the centroid to the reflected point is made (Figure 3.5(c)). The amount of contraction is controlled by a factor  $\beta$  (a negative value of  $\beta$  is used). Finally, if the function value at the reflected point is better than the worst and worse than the next-to-worst point in the simplex, a contraction is made with a positive  $\beta$  value (Figure 3.5(d)). The default scenario is the reflected point itself. The obtained new point replaces the worst point in the simplex and the algorithm continues with the new simplex. This algorithm was originally proposed by Spendley, et al. (1962) and later modified by Nelder and Mead (1965).

### Algorithm

**Step 1** Choose  $\gamma > 1$ ,  $\beta \in (0, 1)$ , and a termination parameter  $\epsilon$ . Create an initial simplex<sup>3</sup>.

**Step 2** Find  $x_h$  (the worst point),  $x_l$  (the best point), and  $x_g$  (next to the worst point). Calculate

$$x_c = \frac{1}{N} \sum_{i=1, i \neq h}^{N+1} x_i.$$

**Step 3** Calculate the reflected point  $x_r = 2x_c - x_h$ . Set  $x_{\text{new}} = x_r$ .

If  $f(x_r) < f(x_l)$ , set  $x_{\text{new}} = (1 + \gamma)x_c - \gamma x_h$  (expansion);

Else if  $f(x_r) \geq f(x_h)$ , set  $x_{\text{new}} = (1 - \beta)x_c + \beta x_h$  (contraction);

Else if  $f(x_g) < f(x_r) < f(x_h)$ , set  $x_{\text{new}} = (1 + \beta)x_c - \beta x_h$  (contraction).

Calculate  $f(x_{\text{new}})$  and replace  $x_h$  by  $x_{\text{new}}$ .

**Step 4** If  $\left\{ \sum_{i=1}^{N+1} \frac{(f(x_i) - f(x_c))^2}{N+1} \right\}^{1/2} \leq \epsilon$ , **Terminate**;

Else go to Step 2.

Any other termination criteria may also be used. The performance of the above algorithm depends in the values of  $\beta$  and  $\gamma$ . If a large value of  $\gamma$  or  $1/\beta$  is used, the approach to the optimum point may be faster, but the convergence to the optimum point may be difficult. On the other hand, smaller values of  $\gamma$  or  $1/\beta$  may require more function evaluations to converge near the optimum point. The recommended values for parameters are  $\gamma \approx 2.0$  and  $|\beta| \approx 0.5$ .

### EXERCISE 3.3.2

Consider, as an example, the Himmelblau function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** We choose an initial simplex of three points as follows:  $x^{(1)} = (0, 0)^T$ ,  $x^{(2)} = (2, 0)^T$ , and  $x^{(3)} = (1, 1)^T$ . The corresponding function values are  $f(x^{(1)}) = 170$ ,  $f(x^{(2)}) = 74$ , and  $f(x^{(3)}) = 106$ , respectively. We also set  $\gamma = 1.5$ ,  $\beta = 0.5$ , and  $\epsilon = 10^{-3}$ .

---

<sup>3</sup>One of the ways to create a simplex is to choose a base point  $x^0$  and a scale factor  $C$ . Then  $(N + 1)$  points are  $x^{(0)}$  and for  $i, j = 1, 2, \dots, N$

$$x_j^{(i)} = \begin{cases} x_j^{(0)} + C & \text{if } j = i \\ x_j^{(0)} + C\Delta & \text{otherwise,} \end{cases} \quad \text{where } \Delta = \begin{cases} 0.25 & \text{if } N = 3 \\ \frac{\sqrt{N+1}-2}{N-3} & \text{otherwise.} \end{cases}$$

Any other initial simplex may be used, but care should be taken not to choose a simplex with a zero hypervolume.

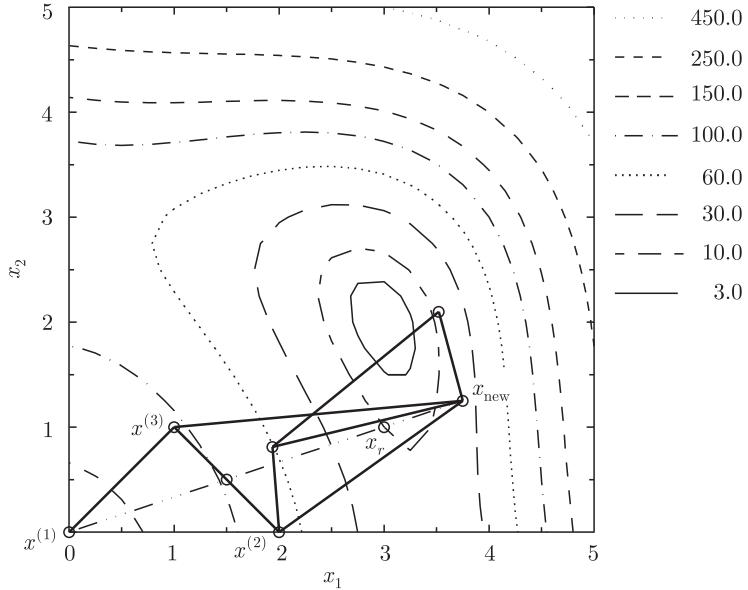
**Step 2** The worst point is  $x_h = x^{(1)}$ , the best point is  $x_l = x^{(2)}$ , and next to the worst point is  $x_g = x^{(3)}$ . Thus, we calculate the centroid of  $x^{(2)}$  and  $x^{(3)}$  as follows:

$$x_c = (x^{(2)} + x^{(3)})/2 = (1.5, 0.5)^T.$$

**Step 3** We compute the reflected point,  $x_r = 2x_c - x_h = (3, 1)^T$ . The corresponding function value is  $f(x_r) = 10$ . Since  $f(x_r) = 10$ , which is less than  $f(x_l) = 74$ , we expand the simplex to find a new point

$$x_{\text{new}} = (1 + 1.5)x_c - 1.5x_h = (3.75, 1.25)^T.$$

The function value at this point is 21.440. Thus, the new simplex is  $x^{(1)} = (3.75, 1.25)^T$ ,  $x^{(2)} = (2, 0)^T$ , and  $x^{(3)} = (1, 1)^T$ . The new simplex is shown in Figure 3.6 ( $x^{(2)}$ ,  $x^{(3)}$  and  $x_{\text{new}}$ ). It is interesting to note that even though the reflected point  $x_r$  is better than the new point, the basic simplex search algorithm does not allow this point in the new simplex.



**Figure 3.6** Three iterations of the simplex search method shown on a contour plot of Himmelblau's function.

**Step 4** We compute the quantity  $Q$  for termination:  $Q = 44.86$ , which is greater than  $\epsilon$ . Thus, we move to Step 2. This completes one iteration of the simplex search method.

**Step 2** In the new simplex, the worst point is  $x_h = x^{(3)}$ , the best point is  $x_l = x^{(1)}$ , and  $x_g = x^{(2)}$ . Thus, the centroid is

$$x_c = (x^{(1)} + x^{(2)})/2 = (2.875, 0.625)^T.$$

The function value at this point is  $f(x_c) = 7.458$ .

**Step 3** The reflected point is  $x_r = 2x_c - x_h = (4.75, 0.25)^T$  and the function value is 144.32. Since  $f(x_r) > f(x_h)$ , we contract the simplex and find a new point

$$x_{\text{new}} = (1 - 0.5)x_c + 0.5x_h = (1.937, 0.812)^T.$$

The corresponding function value is 60.772. Thus, the new simplex is as follows:

$$x^{(1)} = (3.75, 1.25)^T, \quad x^{(2)} = (2, 0)^T, \quad x^{(3)} = (1.937, 0.812)^T.$$

Figure 3.6 also shows the new simplex found at this iteration.

**Step 4** Since  $Q = 94.91 > \epsilon$ , we proceed to Step 2 with the new simplex.

At every iteration, the worst point in the simplex is replaced by a new point. By reflecting the worst point about the centroid of other points, the algorithm works by finding a pseudo search direction (from the worst point towards the centroid of other points in the simplex) at every iteration. If the primary aim is to constitute the search along a direction, algorithms can be designed to work explicitly with directions, which is discussed in Sections 3.3.3 and 3.3.4.

### 3.3.3 Hooke-Jeeves Pattern Search Method

The pattern search method works by creating a set of search directions iteratively. The created search directions should be such that they completely span the search space. In other words, they should be such that starting from any point in the search space any other point in the search space can be reached by traversing along these search directions only. In a  $N$ -dimensional problem, this requires at least  $N$  linearly independent search directions. For example, in a two-variable function, at least two search directions are required to go from any one point to any other point. Among many possible combinations of  $N$  search directions, some combinations may be able to reach the destination faster (with lesser iterations), and some may require more iterations.

In the Hooke-Jeeves method, a combination of exploratory moves and heuristic pattern moves is made iteratively. An exploratory move is performed in the vicinity of the current point systematically to find the best point around the current point. Thereafter, two such points are used to make a pattern move. We describe each of these moves in the following paragraphs:

#### Exploratory move

Assume that the current solution (the base point) is denoted by  $x^c$ . Assume also that the variable  $x_i^c$  is perturbed by  $\Delta_i$ . Set  $i = 1$  and  $x = x^c$ .

**Step 1** Calculate  $f = f(x)$ ,  $f^+ = f(x_i + \Delta_i)$  and  $f^- = f(x_i - \Delta_i)$ .

**Step 2** Find  $f_{\min} = \min(f, f^+, f^-)$ . Set  $x$  corresponds to  $f_{\min}$ .

**Step 3** Is  $i = N$ ? If no, set  $i = i + 1$  and go to Step 1;  
Else  $x$  is the result and go to Step 4.

**Step 4** If  $x \neq x^c$ , **success**; Else **failure**.

In the exploratory move, the current point is perturbed in positive and negative directions along each variable one at a time and the best point is recorded. The current point is changed to the best point at the end of each variable perturbation. If the point found at the end of all variable perturbations is different than the original point, the exploratory move is a *success*, otherwise the exploratory move is a *failure*. In any case, the best point is considered to be the outcome of the exploratory move.

### Pattern move

A new point is found by jumping from the current best point  $x^c$  along a direction connecting the previous best point  $x^{(k-1)}$  and the current base point  $x^{(k)}$  as follows:

$$x_p^{(k+1)} = x^{(k)} + (x^{(k)} - x^{(k-1)}).$$

The Hooke-Jeeves method comprises of an iterative application of an exploratory move in the locality of the current point and a subsequent jump using the pattern move. If the pattern move does not take the solution to a better region, the pattern move is not accepted and the extent of the exploratory search is reduced. The algorithm works as follows:

### Algorithm

**Step 1** Choose a starting point  $x^{(0)}$ , variable increments  $\Delta_i$  ( $i = 1, 2, \dots, N$ ), a step reduction factor  $\alpha > 1$ , and a termination parameter,  $\epsilon$ . Set  $k = 0$ .

**Step 2** Perform an exploratory move with  $x^{(k)}$  as the base point. Say  $x$  is the outcome of the exploratory move. If the exploratory move is a success, set  $x^{(k+1)} = x$  and go to Step 4;

Else go to Step 3.

**Step 3** Is  $\|\Delta\| < \epsilon$ ? If yes, **Terminate**;  
Else set  $\Delta_i = \Delta_i / \alpha$  for  $i = 1, 2, \dots, N$  and go to Step 2.

**Step 4** Set  $k = k + 1$  and perform the pattern move:

$$x_p^{(k+1)} = x^{(k)} + (x^{(k)} - x^{(k-1)}).$$

**Step 5** Perform another exploratory move using  $x_p^{(k+1)}$  as the base point.  
Let the result be  $x^{(k+1)}$ .

**Step 6** Is  $f(x^{(k+1)}) < f(x^{(k)})$ ? If yes, go to Step 4;  
Else go to Step 3.

The search strategy is simple and straightforward. The algorithm requires less storage for variables; only two points ( $x^{(k)}$  and  $x^{(k+1)}$ ) need to be stored at any iteration. The numerical calculations involved in the process are also simple. But, since the search largely depends on the moves along the coordinate directions ( $x_1$ ,  $x_2$ , and so on) during the exploratory move, the algorithm may prematurely converge to a wrong solution, especially in the case of functions with highly nonlinear interactions among variables. The algorithm may also get stuck in the loop of generating exploratory moves either between Steps 5 and 6 or between Steps 2 and 3. Another feature of this algorithm is that it terminates only by exhaustively searching the vicinity of the converged point. This requires a large number of function evaluations for convergence to a solution with a reasonable degree of accuracy. The convergence to the optimum point depends on the parameter  $\alpha$ . A value  $\alpha = 2$  is recommended.

The working principle of the algorithm can be better understood through the following hand-simulation on a numerical exercise problem.

### EXERCISE 3.3.3

Consider the Himmelblau function again:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** Let us choose the initial point to be  $x^{(0)} = (0, 0)^T$ , the increment vector  $\Delta = (0.5, 0.5)^T$ , and reduction factor  $\alpha = 2$ . We also set a termination parameter  $\epsilon = 10^{-3}$  and an iteration counter  $k = 0$ .

**Step 2** We perform an iteration of exploratory move using  $x^{(0)}$  as the base point. Thus, we set  $x = x^c = (0, 0)^T$  and  $i = 1$ . The steps of the exploratory move are worked out in the following:

**Step 1** We first explore the vicinity of the variable  $x_1$ . We calculate the function values at three points:

$$(x_1^{(0)} + \Delta_1, x_2^{(0)})^T = (0.5, 0.0)^T, \quad f^+ = f((0.5, 0)^T) = 157.81,$$

$$x^{(0)} = (0.0, 0.0)^T, \quad f = f((0, 0)^T) = 170,$$

$$(x_1^{(0)} - \Delta_1, x_2^{(0)})^T = (-0.5, 0.0)^T, \quad f^- = 171.81.$$

**Step 2** The minimum of the above three function values is  $f_{\min} = 157.81$ , and the corresponding point is  $x = (0.5, 0.0)^T$ .

**Step 3** At this iteration  $i \neq 2$  (not all variables are explored yet). Thus, we increment the counter  $i = 2$  and explore the second variable  $x_2$ . This completes one iteration of the exploratory move.

**Step 1** Note that at this point, the base point is  $(0.5, 0)^T$ . We explore variable  $x_2$  at this point and calculate function values at the following three points:

$$f^+ = f((0.5, 0.5)^T) = 144.12,$$

$$f = f((0.5, 0.0)^T) = 157.81,$$

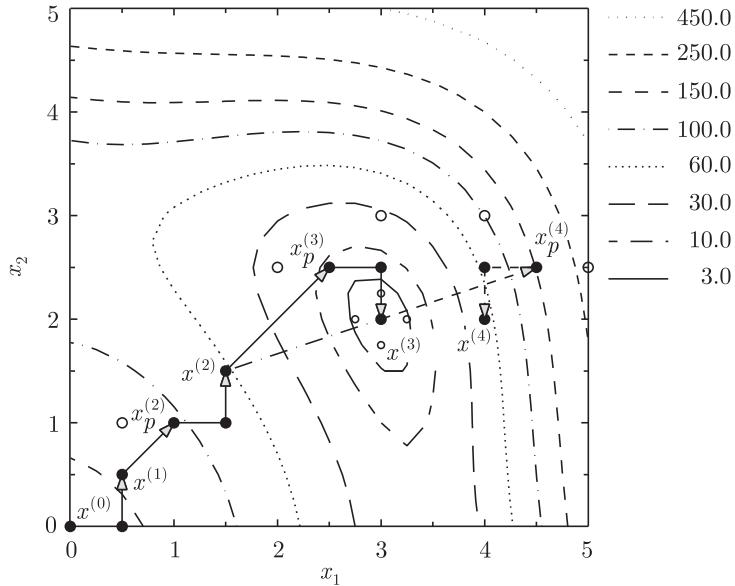
$$f^- = f((0.5, -0.5)^T) = 165.62.$$

**Step 2** Here,  $f_{\min} = 144.12$  and the corresponding point is  $x = (0.5, 0.5)^T$ .

**Step 3** At this step  $i = 2$  and we move to Step 4 of the exploratory move.

**Step 4** Since  $x \neq x^c$ , the exploratory move is a success and we set  $x = (0.5, 0.5)^T$ .

Since the exploratory move is a success, we set  $x^{(1)} = x = (0.5, 0.5)^T$  and move to Step 4. The successful points in the exploratory move of the Hooke-Jeeves algorithm are marked with a filled circle in Figure 3.7.



**Figure 3.7** Four iterations of Hooke-Jeeves search method shown on a contour plot of the Himmelblau function.

**Step 4** We set  $k = 1$  and perform a pattern move:

$$x_p^{(2)} = x^{(1)} + (x^{(1)} - x^{(0)}) = 2(0.5, 0.5)^T - (0, 0)^T = (1, 1)^T.$$

**Step 5** We perform another exploratory move using  $x_p^{(2)}$  as the base point. After performing the exploratory move as before, we observe that the search

is a success and the new point is  $x = (1.5, 1.5)^T$ . We set the new point  $x^{(2)} = x = (1.5, 1.5)^T$ .

**Step 6** We observe that  $f(x^{(2)}) = 63.12$ , which is smaller than  $f(x^{(1)}) = 144.12$ . Thus, we proceed to Step 4 to perform another pattern move. This completes one iteration of the Hooke-Jeeves method.

**Step 4** We set  $k = 2$  and create a new point

$$x_p^{(3)} = 2x^{(2)} - x^{(1)} = (2.5, 2.5)^T.$$

It is interesting to note here that since  $x^{(2)}$  is better than  $x^{(1)}$ , a jump along the direction  $(x^{(2)} - x^{(1)})$  is made. This jump takes the search closer towards the true minimum.

**Step 5** We perform another exploratory move locally to find out if there is any better point around the new point. After performing an exploratory move on both variables, it is found that the search is a success and the new point is  $x^{(3)} = (3.0, 2.0)^T$ . This new point is incidentally the true minimum point. In this example, the chosen initial point and the increment vector happen to be such that in two iterations of the Hooke-Jeeves algorithm the minimum point is obtained. But, in general, more iterations may be required. An interesting point to note that even though the minimum point is found in two iterations, the algorithm has no way of knowing whether the optimum is reached or not. The algorithm proceeds until the norm of the increment vector  $\Delta$  is small. We continue to perform two more iterations to see how the algorithm may finally terminate at the minimum point.

**Step 6** Calculating the function value at the new point, we observe that  $f(x^{(3)}) = 0 < f(x^{(2)}) = 63.12$ . Thus, we move to Step 4.

**Step 4** The iteration counter  $k = 3$  and the new point is  $x_p^{(4)} = 2x^{(3)} - x^{(2)} = (4.5, 2.5)^T$ .

**Step 5** By performing an exploratory move with  $x_p^{(4)}$  as the base point, we find that the search is a success and  $x = (4.0, 2.0)^T$ . Thus, we set  $x^{(4)} = (4.0, 2.0)^T$ .

**Step 6** The function value at this point is  $f(x^{(4)}) = 50$ , which is larger than  $f(x^{(3)}) = 0$ . Thus, we move to Step 3.

**Step 3** Since  $\|\Delta\| = 0.5 \not< \epsilon$ , we reduce the increment vector

$$\Delta = (-0.5, 0.5)^T / 2 = (0.25, 0.25)^T$$

and proceed to Step 2 to perform the next iteration.

**Step 2** We perform an exploratory move with  $x^{(3)} = (3.0, 2.0)^T$  as the current point. The exploratory move on both variables is a failure and we obtain  $x = (3.0, 2.0)^T$ . Thus we proceed to Step 3.

**Step 3** Since  $\|\Delta\|$  is not small, we reduce the increment vector by  $\alpha$  again and move to Step 2. The new increment vector is  $\Delta = (0.125, 0.125)^T$ .

The algorithm now continues with Steps 2 and 3 until  $\|\Delta\|$  is smaller than the termination factor  $\epsilon$ . Thus, the final solution is  $x^* = (3.0, 2.0)^T$  with a function value  $f(x^*) = 0$ . Figure 3.7 shows the intermediate points obtained using the Hooke-Jeeves pattern search algorithm.

### 3.3.4 Powell's Conjugate Direction Method

The conjugate direction method is probably the most successful and popular direct search method used in many engineering optimization problems. It uses a history of previous solutions to create new search directions. Unlike previous methods, this method has a convergence proof for quadratic objective functions, even though many non-quadratic functions have been successfully solved using this method.

The basic idea is to create a set of  $N$  linearly independent search directions and perform a series of unidirectional searches along each of these search directions, starting each time from the previous best point. This procedure guarantees to find the minimum of a quadratic function by one pass of  $N$  unidirectional searches along each search direction. In other functions, more than one pass of  $N$  unidirectional searches are necessary. The algorithm is designed on the basis of solving a quadratic function and has the following property.

#### Parallel subspace property

Given a quadratic function  $q(x) = A + B^T x + \frac{1}{2}x^T C x$  of two variables (where  $A$  is a scalar quantity,  $B$  is a vector, and  $C$  is a  $2 \times 2$  matrix), two arbitrary but distinct points  $x^{(1)}$  and  $x^{(2)}$ , and a direction  $d$ .

If  $y^{(1)}$  is the solution to the problem

$$\text{minimize } q(x^{(1)} + \lambda d)$$

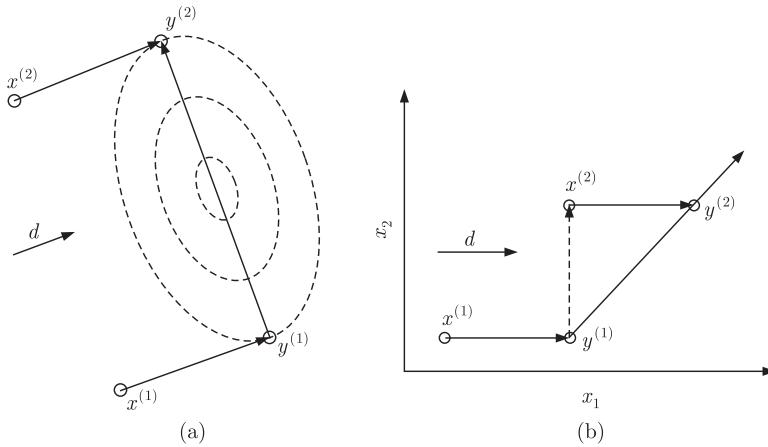
and  $y^{(2)}$  is the solution to the problem

$$\text{minimize } q(x^{(2)} + \lambda d)$$

then the direction  $(y^{(2)} - y^{(1)})$  is conjugate to  $d$  or, in other words, the quantity  $(y^{(2)} - y^{(1)})^T C d$  is zero.

Thus, if two arbitrary points  $x^{(1)}$  and  $x^{(2)}$  and an arbitrary search direction  $d$  are chosen, two unidirectional searches, one from each point will create two points  $y^{(1)}$  and  $y^{(2)}$ . For quadratic functions, we can say that the minimum of the function lies on the line joining the points  $y^{(1)}$  and  $y^{(2)}$ , as depicted in Figure 3.8(a). The vector  $(y^{(2)} - y^{(1)})$  forms a conjugate direction with the original direction vector  $d$ .

Instead of using two points  $(x^{(1)}$  and  $x^{(2)}$ ) and a direction vector ( $d$ ) to create one pair of conjugate directions, one point  $(x^{(1)})$  and both coordinate



**Figure 3.8** An illustration of the parallel subspace property with two arbitrary points and an arbitrary search direction in (a). The same can also be achieved from one point and two coordinate points in (b).

directions  $((1, 0)^T$  and  $(0, 1)^T$ ) can be used to create a pair of conjugate directions ( $d$  and  $(y^{(2)} - y^{(1)})$ ) (Figure 3.8(b)). The point  $y^{(1)}$  is obtained by performing a unidirectional search along  $(1, 0)^T$  from the point  $x^{(1)}$ . Then, the point  $x^{(2)}$  is obtained by performing a unidirectional search along  $(0, 1)^T$  from  $y^{(1)}$  and finally the point  $y^{(2)}$  is found by a unidirectional search along the direction  $(1, 0)^T$  from the point  $x^{(2)}$ . By comparing Figures 3.8(a) and 3.8(b), we notice that both figures follow the parallel subspace property. The former approach requires two unidirectional searches to find a pair of conjugate directions, whereas the latter approach requires three unidirectional searches. For a quadratic function, the minimum lies in the direction  $(y^{(2)} - y^{(1)})$ , but for higher-order polynomials the true minimum may not lie in the above direction. Thus, in the case of quadratic function, four unidirectional searches will find the minimum point and in higher-order polynomials more than four unidirectional searches may be necessary. In the latter case, a few iterations of this procedure are required to find the true minimum point. This concept of parallel subspace property can also be extended to higher dimensions.

#### Extended parallel subspace property

Let us assume that the point  $y^{(1)}$  is found after unidirectional searches along each of  $m (< N)$  conjugate directions from a chosen point  $x^{(1)}$  and, similarly, the point  $y^{(2)}$  is found after unidirectional searches along each of  $m$  conjugate directions from another point  $x^{(2)}$ . The vector  $(y^{(2)} - y^{(1)})$  is the conjugate to all  $m$  search directions.

The extended parallel subspace property can also be used starting from one point  $x^{(1)}$  and  $N$  coordinate directions. The point  $y^{(1)}$  can be found after a unidirectional search along one of the search directions (say

$e^{(1)} = (1, 0, \dots, 0)^T$ . Thereafter the point  $y^{(2)}$  is found after searches in  $N$  coordinate directions ( $e^{(i)}$ ,  $i = 2, 3, \dots, N, 1$ ) with  $e^{(1)}$  being the final search direction. Then the vector  $(y^{(2)} - y^{(1)})$  is conjugate to the search direction  $e^{(1)}$ . The coordinate direction  $e^{(N)}$  can now be replaced with the new search direction  $(y^{(2)} - y^{(1)})$  and the same procedure can be followed starting from  $e^{(2)}$ . With this property, we now present the algorithm:

#### Algorithm

**Step 1** Choose a starting point  $x^{(0)}$  and a set of  $N$  linearly independent directions; possibly  $s^{(i)} = e^{(i)}$  for  $i = 1, 2, \dots, N$ .

**Step 2** Minimize along  $N$  unidirectional search directions using the previous minimum point to begin the next search. Begin with the search direction  $s^{(1)}$ . and end with  $s^{(N)}$ . Thereafter, perform another unidirectional search along  $s^{(1)}$ .

**Step 3** Form a new conjugate direction  $d$  using the extended parallel subspace property.

**Step 4** If  $\|d\|$  is small or search directions are linearly dependent, **Terminate**;

Else replace  $s^{(j)} = s^{(j-1)}$  for all  $j = N, N-1, \dots, 2$ . Set  $s^{(1)} = d/\|d\|$  and go to Step 2.

If the function is quadratic, exactly  $(N-1)$  loops through Steps 2 to 4 is required. Since in every iteration of the above algorithm exactly  $(N+1)$  unidirectional searches are necessary, a total of  $(N-1) \times (N+1)$  or  $(N^2-1)$  unidirectional searches are necessary to find  $N$  conjugate directions. Thereafter, one final unidirectional search is necessary to obtain the minimum point. Thus, in order to find the minimum of a quadratic objective function, the conjugate direction method requires a total of  $N^2$  unidirectional searches. For other functions, more loops of the above algorithm may be required. One difficulty with this algorithm is that since unidirectional searches are carried out numerically by using a single-variable search method, the computation of the minimum for unidirectional searches may not be exact. Thus, the resulting directions may not be exactly conjugate to each other. To calculate the extent of deviation, linear independence of the conjugate directions is usually checked. If the search directions are not found to be linearly independent, a completely new set of search directions (possibly conjugate to each other) may be created at the current point. To make the implementation simpler, the coordinate directions can be used again as search directions at the current point.

#### EXERCISE 3.3.4

Consider again the Himmelblau function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** We begin with a point  $x^{(0)} = (0, 4)^T$ . We assume initial search directions as  $s^{(1)} = (1, 0)^T$  and  $s^{(2)} = (0, 1)^T$ .

**Step 2** We first find the minimum point along the search direction  $s^{(1)}$ . Any point along that direction can be written as  $x^p = x^{(0)} + \alpha s^{(1)}$ , where  $\alpha$  is a scalar quantity expressing the distance of the point  $x^p$  from  $x^{(0)}$ . Thus, the point  $x^p$  can be written as  $x^p = (\alpha, 4)^T$ . Now the two-variable function  $f(x_1, x_2)$  can be expressed in terms of one variable  $\alpha$  as

$$F(\alpha) = (\alpha^2 - 7)^2 + (\alpha + 9)^2,$$

which represents the function value of any point along the direction  $s^{(1)}$  and passing through  $x^{(0)}$ . Since we are looking for the point for which the function value is minimum, we may differentiate the above expression with respect to  $\alpha$  and equate to zero. But in any arbitrary problem, it may not be possible to write an explicit expression of the single-variable function  $F(\alpha)$  and differentiate. In those cases, the function  $F(\alpha)$  can be obtained by substituting each variable  $x_i$  by  $x_i^p$ . Thereafter, any single-variable optimization methods, as described in Chapter 2, can be used to find the minimum point. The first task is to bracket the minimum and then the subsequent task is to find the minimum point. Here, we could have found the exact minimum solution by differentiating the single-variable function  $F(\alpha)$  with respect to  $\alpha$  and then equating the term to zero, but we follow the more generic procedure of numerical differentiation, a method which will be used in many real-world optimization problems. Using the bounding phase method in the above problem we find that the minimum is bracketed in the interval  $(1, 4)$  and using the golden section search we obtain the minimum  $\alpha^* = 2.083$  with three decimal places of accuracy<sup>4</sup>. Thus,  $x^{(1)} = (2.083, 4.000)^T$ . The above procedure of obtaining the best point along a search direction is also described in Section 3.2.

Similarly, we find the minimum point along the second search direction  $s^{(2)}$  from the point  $x^{(1)}$ . A general point on that line is

$$x(\alpha) = x^{(1)} + \alpha s^{(2)} = (2.083, (4 + \alpha))^T.$$

The optimum point found using a combined application of the bounding phase and the golden section search method is  $\alpha^* = -1.592$  and the corresponding point is  $x^{(2)} = (2.083, 2.408)^T$ .

From the point  $x^{(2)}$ , we perform a final unidirectional search along the first search direction  $s^{(1)}$  and obtain the minimum point  $x^{(3)} = (2.881, 2.408)^T$ .

**Step 3** According to the parallel subspace property, we find the new conjugate direction

$$d = x^{(3)} - x^{(1)} = (2.881, 2.408)^T - (2.083, 4.000)^T = (0.798, -1.592)^T.$$

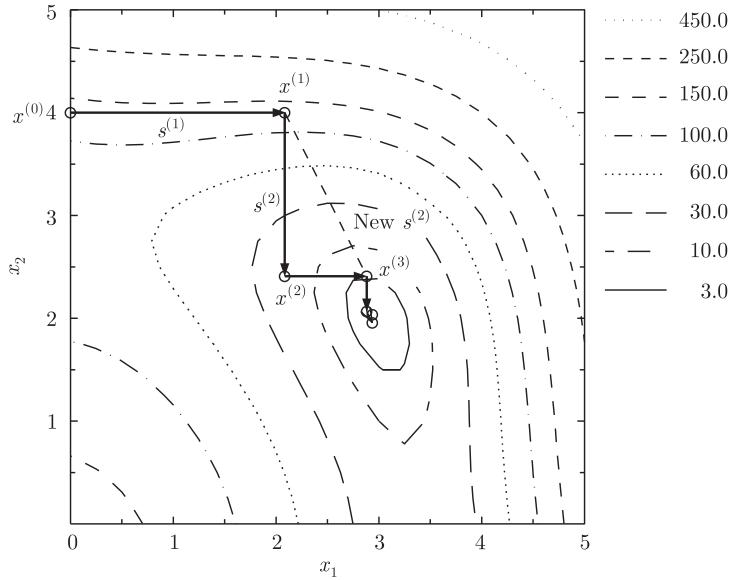
---

<sup>4</sup>FORTRAN codes presented in Chapter 2 are used to find these solutions.

**Step 4** The magnitude of search vector  $d$  is not small. Thus, the new conjugate search directions are

$$\begin{aligned}s^{(2)} &= (1, 0)^T, \\ s^{(1)} &= (0.798, -1.592)^T / \|(0.798, -1.592)^T\| \\ &= (0.448, -0.894)^T.\end{aligned}$$

This completes one iteration of Powell's conjugate direction method. Figure 3.9 shows the new conjugate direction on a contour plot of the objective function. With these new search directions we now proceed to Step 2.



**Figure 3.9** Two iterations of Powell's conjugate direction method.

**Step 2** A single-variable minimization along the search direction  $s^{(1)}$  from the point  $x^{(3)} = (2.881, 2.408)^T$  results in the new point  $x^{(4)} = (3.063, 2.045)^T$ . If the objective function were a quadratic function, we would have achieved that the optimum point at this step. However, it is interesting to note that the solution  $x^{(4)}$  is close to the true minimum of the function.

One more unidirectional search along  $s^{(2)}$  from the point  $x^{(4)}$  results in the point  $x^{(5)} = (2.988, 2.045)^T$ . Another minimization along  $s^{(1)}$  results in  $x^{(6)} = (3.008, 2.006)^T$  ( $f(x^{(6)}) = 0.004$ ).

**Step 3** The new conjugate direction is

$$d = (x^{(6)} - x^{(4)}) = (0.055, -0.039)^T.$$

The unit vector along this direction is  $(0.816, -0.578)^T$ .

**Step 4** The new pair of conjugate search directions are  $s^{(1)} = (0.448, -0.894)^T$  and  $s^{(2)} = (0.055, -0.039)^T$ , respectively. The search direction  $d$  (before normalizing) may be considered to be small and therefore the algorithm may terminate.

We observe that in one iteration of Step 2,  $(N + 1)$  unidirectional searches are necessary. Thus, computationally this method may be expensive. In terms of the storage requirement, the algorithm has to store  $(N + 1)$  points and  $N$  search directions at any stage of the iteration.

### 3.4 Gradient-based Methods

Direct search methods described in Section 3.3 require many function evaluations to converge to the minimum point. Gradient-based methods discussed in this section exploit the derivative information of the function and are usually faster search methods. Since these methods use gradient information and since the objective functions in many engineering optimization problems are not differentiable, they cannot be applied directly to many engineering design problems. For the same reason, the gradient-based methods cannot be applied to problems where the objective function is discrete or discontinuous or the variables are discrete (like in integer programming problems described in Chapter 5). On the contrary, in problems where the derivative information is easily available, gradient-based methods are very efficient. However, the concept of gradients in understanding the working principle of the optimization algorithms is so intricate that gradient-based methods are mostly used in engineering design problems by calculating the derivatives numerically.

Some algorithms described in this section require only the first derivative of the objective function and some algorithms require both the first and the second-order derivatives. The derivatives can be obtained by numerical computation of objective functions only at two or three neighbouring points. The first and the second-order derivatives are calculated based on the central difference technique:

$$\frac{\partial f(x)}{\partial x_i} \Big|_{x^{(t)}} = \left( f(x_i^{(t)} + \Delta x_i^{(t)}) - f(x_i^{(t)} - \Delta x_i^{(t)}) \right) / (2\Delta x_i^{(t)}), \quad (3.4)$$

$$\frac{\partial^2 f(x)}{\partial^2 x_i^2} \Big|_{x^{(t)}} = \left( f(x_i^{(t)} + \Delta x_i^{(t)}) - 2f(x_i^{(t)}) + f(x_i^{(t)} - \Delta x_i^{(t)}) \right) / (\Delta x_i^{(t)})^2, \quad (3.5)$$

$$\begin{aligned} \frac{\partial^2 f(x)}{\partial x_i \partial x_j} \Big|_{x^{(t)}} &= \left[ f(x_i^{(t)} + \Delta x_i^{(t)}, x_j^{(t)} + \Delta x_j^{(t)}) - f(x_i^{(t)} + \Delta x_i^{(t)}, x_j^{(t)} - \Delta x_j^{(t)}) \right. \\ &\quad \left. - f(x_i^{(t)} - \Delta x_i^{(t)}, x_j^{(t)} + \Delta x_j^{(t)}) \right. \\ &\quad \left. + f(x_i^{(t)} - \Delta x_i^{(t)}, x_j^{(t)} - \Delta x_j^{(t)}) \right] / (4\Delta x_i^{(t)} \Delta x_j^{(t)}). \end{aligned} \quad (3.6)$$

The quantity  $f(x_i^{(t)} + \Delta x_i^{(t)})$  represents the function value at the point  $(x_1^{(t)}, \dots, x_i^{(t)} + \Delta x_i^{(t)}, \dots, x_N^{(t)})^T$ , a point obtained by perturbing the variable  $x_i$  only. The quantity  $f(x_i^{(t)} + \Delta x_i^{(t)}, x_j^{(t)} + \Delta x_j^{(t)})$  represents the function value at the point  $(x_1^{(t)}, \dots, x_i^{(t)} + \Delta x_i^{(t)}, \dots, x_j^{(t)} + \Delta x_j^{(t)}, \dots, x_N^{(t)})^T$ , a point obtained by perturbing the variables  $x_i$  and  $x_j$  only.

The computation of the first derivative with respect to each variable requires two function evaluations, thus totaling  $2N$  function evaluations for the complete first derivative vector. The computation of the second derivative  $\partial^2 f / \partial x_i^2$  requires three function evaluations, but the second-order partial derivative  $\partial^2 f / (\partial x_i \partial x_j)$  requires four function evaluations as given in Equation (3.6). Thus, the computation of Hessian matrix requires  $(2N^2 + 1)$  function evaluations (assuming the symmetry of the matrix). For example, in two-variable functions the first and the second derivatives are computed as

$$\frac{\partial f(x)}{\partial x_1} \Big|_{x^{(t)}} = \left( f(x_1^{(t)} + \Delta x_1^{(1)}, x^{(2)}) - f(x_1^{(t)} - \Delta x_1^{(1)}, x^{(2)}) \right) / (2\Delta x_1^{(t)}), \quad (3.7)$$

$$\begin{aligned} \frac{\partial^2 f(x)}{\partial x_1^2} \Big|_{x^{(t)}} &= \left[ f(x_1^{(t)} + \Delta x_1^{(t)}, x_2^{(t)}) - 2f(x_1^{(t)}, x_2^{(t)}) \right. \\ &\quad \left. + f(x_1^{(t)} - \Delta x_1^{(t)}, x_2^{(t)}) \right] / (\Delta x_1^{(t)})^2, \end{aligned} \quad (3.8)$$

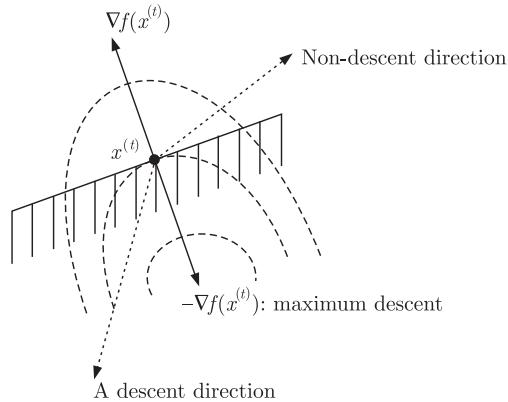
$$\begin{aligned} \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} \Big|_{x^{(t)}} &= \left[ f(x_1^{(t)} + \Delta x_1^{(t)}, x_2^{(t)} + \Delta x_2^{(t)}) - f(x_1^{(t)} + \Delta x_1^{(t)}, x_2^{(t)} - \Delta x_2^{(t)}) \right. \\ &\quad \left. - f(x_1^{(t)} - \Delta x_1^{(t)}, x_2^{(t)} + \Delta x_2^{(t)}) \right. \\ &\quad \left. + f(x_1^{(t)} - \Delta x_1^{(t)}, x_2^{(t)} - \Delta x_2^{(t)}) \right] / (4\Delta x_1^{(t)} \Delta x_2^{(t)}). \end{aligned} \quad (3.9)$$

In the above computations, the first-order derivative requires four function evaluations and the Hessian matrix requires nine function evaluations. By definition, the first derivative  $\nabla f(x^{(t)})$  at any point  $x^{(t)}$  represents the direction of the maximum increase of the function value. Thus, if we are interested in finding a point with the minimum function value, ideally we should be searching along the opposite to the first derivative direction, that is, we should search along  $-\nabla f(x^{(t)})$  direction (see Figure 3.10). Any search direction  $d^{(t)}$  in the hatched region in Figure 3.10 would have smaller function value than that at the current point  $x^{(t)}$ . Thus, a search direction  $d^{(t)}$  that satisfies the following relation is a *descent* direction.

### Descent direction

A search direction,  $d^{(t)}$ , is a descent direction at point  $x^{(t)}$  if the condition  $\nabla f(x^{(t)}) \cdot d^{(t)} \leq 0$  is satisfied in the vicinity of the point  $x^{(t)}$ .

The above condition for descent search direction can also be proved by comparing function values at two points along any descent direction



**Figure 3.10** An illustration of a descent direction. Any direction in the hatched region is a descent direction.

$f(x^{(t+1)}) < f(x^{(t)})$  and expanding the expression  $f(x^{(t+1)})$  in Taylor's series up to the linear term:

$$f(x^{(t+1)}) = f(x^{(t)} + \alpha d^{(t)}) = f(x^{(t)}) + \alpha \nabla f(x^{(t)}) \cdot d^{(t)}. \quad (3.10)$$

The magnitude of the vector  $\nabla f(x^{(t)}) \cdot d^{(t)}$  for a descent direction  $d^{(t)}$  specifies how descent the search direction is. For example, if  $d_2^{(t)} = -\nabla f(x^{(t)})$  is used, the quantity  $\nabla f(x^{(t)}) \cdot d^{(t)}$  is maximally negative. Thus, the search direction  $-\nabla f(x^{(t)})$  is called the steepest descent direction.

#### EXERCISE 3.4.1

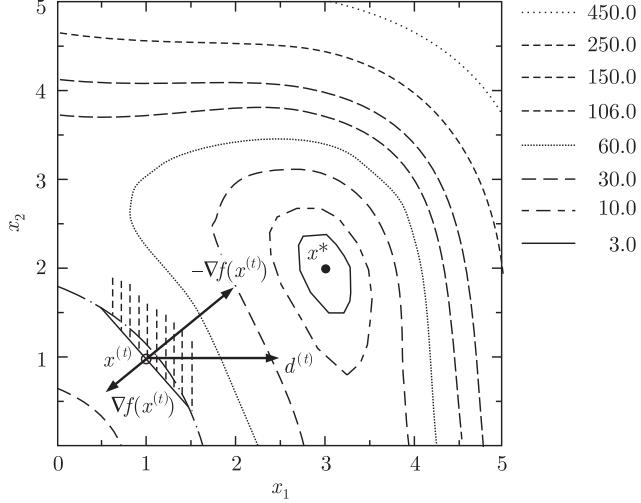
We take an exercise problem to illustrate this concept. Let us consider Himmelblau's function again:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

We would like to determine whether the direction  $d^{(t)} = (1, 0)^T$  at the point  $x^{(t)} = (1, 1)^T$  is a descent direction or not. The point and the direction are shown on a contour plot in Figure 3.11. It is clear from the figure that moving locally along  $d^{(t)}$  from the point  $x^{(t)}$  will reduce the function value. We investigate this aspect by calculating the derivative  $\nabla f(x^{(t)})$  at the point. The derivative, as calculated numerically, is  $\nabla f(x^{(t)}) = (-46, -38)^T$ . Taking the dot product between  $\nabla f(x^{(t)})$  and  $d^{(t)}$ , we obtain

$$\nabla f(x^{(t)})^T d^{(t)} = (-46, -38) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = -46,$$

which is a negative quantity. Thus, the search direction  $d^{(t)} = (1, 0)^T$  is a descent direction. The amount of nonnegativity suggests the extent of descent



**Figure 3.11** The direction  $(1, 0)^T$  is a descent direction, whereas the direction  $-\nabla f(x^{(t)})$  is the steepest descent direction.

in the direction. If the search direction  $d^{(t)} = -\nabla f(x^{(t)}) = (46, 38)^T$  is used, the magnitude of the above dot product becomes

$$(-46, -38)^T \begin{pmatrix} 46 \\ 38 \end{pmatrix} = -3,560.$$

Thus, the direction  $(46, 38)^T$  or  $(0.771, 0.637)^T$  is more descent than  $d^{(t)}$ . In fact, it can be proved that the above direction  $(0.771, 0.637)^T$  is the steepest descent direction at the point  $x^{(t)}$ , as shown in Figure 3.11. It is noteworthy that for any nonlinear function, the steepest descent direction at any point may not exactly pass through the true minimum point. The steepest descent direction is a direction which is a local best direction. It is not guaranteed that moving along the steepest descent direction will always take the search closer to the true minimum point. We shall discuss more about this aspect in Chapter 6.

Most gradient-based methods work by searching along several directions iteratively. The algorithms vary according to the way the search directions are defined. Along each direction,  $s^{(k)}$ , a unidirectional search is performed to locate the minimum. This is performed by first writing a representative point along the direction  $s^{(t)}$  as follows:

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} s^{(k)},$$

where  $\alpha^{(k)}$  is the step length. Since  $x^{(k)}$  and  $s^{(k)}$  are known, the point  $x^{(k+1)}$  can be expressed with only one variable  $\alpha^{(k)}$ . Thus, a unidirectional search on  $\alpha^{(k)}$  can be performed and the new point  $x^{(k+1)}$  can be obtained. Thereafter, the search is continued from the new point along another search direction

$s^{(k+1)}$ . This process continues until the search converges to a local minimum point. If a gradient-based search method is used for unidirectional search, the search can be terminated using the following procedure. By differentiating the expression  $f(x^{(k+1)}) = f(x^{(k)} + \alpha s^{(k)})$  with respect to  $\alpha$  and satisfying the optimality criterion, it can be shown that the minimum of the unidirectional search occurs when

$$\nabla f(x^{(k+1)}) \cdot s^{(k)} = 0.$$

The above criterion can be used to check the termination of the unidirectional search method.

### 3.4.1 Cauchy's (steepest descent) Method

The search direction used in Cauchy's method is the negative of the gradient at any particular point  $x^{(t)}$ :

$$s^{(k)} = -\nabla f(x^{(k)}).$$

Since this direction gives maximum descent in function values, it is also known as the steepest descent method. At every iteration, the derivative is computed at the current point and a unidirectional search is performed in the negative to this derivative direction to find the minimum point along that direction. The minimum point becomes the current point and the search is continued from this point. The algorithm continues until a point having a small enough gradient vector is found. This algorithm guarantees improvement in the function value at every iteration.

#### Algorithm

**Step 1** Choose a maximum number of iterations  $M$  to be performed, an initial point  $x^{(0)}$ , two termination parameters  $\epsilon_1, \epsilon_2$ , and set  $k = 0$ .

**Step 2** Calculate  $\nabla f(x^{(k)})$ , the first derivative at the point  $x^{(k)}$ .

**Step 3** If  $\|\nabla f(x^{(k)})\| \leq \epsilon_1$ , Terminate;

Else if  $k \geq M$ ; Terminate;

Else go to Step 4.

**Step 4** Perform a unidirectional search to find  $\alpha^{(k)}$  using  $\epsilon_2$  such that  $f(x^{(k+1)}) = f(x^{(k)} - \alpha^{(k)} \nabla f(x^{(k)}))$  is minimum. One criterion for termination is when  $|\nabla f(x^{(k+1)}) \cdot \nabla f(x^{(k)})| \leq \epsilon_2$ .

**Step 5** Is  $\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k)}\|} \leq \epsilon_1$ ? If yes, Terminate;

Else set  $k = k + 1$  and go to Step 2.

Since the direction  $s^{(k)} = -\nabla f(x^{(k)})$  is a descent direction, the function value  $f(x^{(k+1)})$  is always smaller than  $f(x^{(k)})$  for positive values of  $\alpha^{(k)}$ .

Cauchy's method works well when  $x^{(0)}$  is far away from  $x^*$ . When the current point is very close to the minimum, the change in the gradient vector is small. Thus, the new point created by the unidirectional search is also close to the current point. This slows the convergence process near the true minimum. Convergence can be made faster by using the second-order derivatives, a method which is discussed in Section 3.4.2.

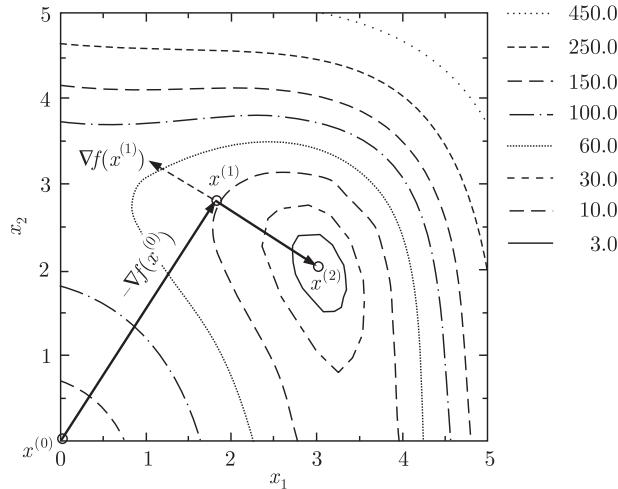
### EXERCISE 3.4.2

Consider the Himmelblau function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** In order to ensure proper convergence, a large value of  $M$  ( $\approx 100$ ) is usually chosen. The choice of  $M$  also depends on the available time and computing resource. Let us choose  $M = 100$ , an initial point  $x^{(0)} = (0, 0)^T$ , and termination parameters  $\epsilon_1 = \epsilon_2 = 10^{-3}$ . We also set  $k = 0$ .

**Step 2** The derivative at  $x^{(0)}$  is first calculated using Equation (3.4) and found to be  $(-14, -22)^T$ , which is identical to the exact derivative at that point (Figure 3.12).



**Figure 3.12** A few iterations of Cauchy's method shown on a contour plot of Himmelblau's function.

**Step 3** The magnitude of the derivative vector is not small and  $k = 0 < M = 100$ . Thus, we do not terminate; rather we proceed to Step 4.

**Step 4** At this step, we need to perform a line search from  $x^{(0)}$  in the direction  $-\nabla f(x^{(0)})$  such that the function value is minimum. Along that direction, any point can be expressed by fixing a value for the parameter  $\alpha^{(0)}$  in the equation:

$$x = x^{(0)} - \alpha^{(0)} \nabla f(x^{(0)}) = (14\alpha^{(0)}, 22\alpha^{(0)})^T.$$

Using the golden section search (Section 2.3.3) in the interval<sup>5</sup>  $(0, 1)$ , we obtain  $\alpha^{(0)*} = 0.127$ . Thus, the minimum point along the search direction is  $x^{(1)} = (1.788, 2.810)^T$ .

**Step 5** Since,  $x^{(1)}$  and  $x^{(0)}$  are quite different, we do not terminate; rather we move back to Step 2. This completes one iteration of Cauchy's method. The total number of function evaluations required in this iteration is equal to 30.

**Step 2** The derivative vector at this point, computed numerically, is  $(-30.707, 18.803)^T$ .

**Step 3** This magnitude of the derivative vector is not smaller than  $\epsilon_1$ . Thus, we continue with Step 4.

**Step 4** Another unidirectional search along  $(30.707, -18.803)^T$  from the point  $x^{(1)} = (1.788, 2.810)^T$  using the golden section search finds the new point  $x^{(2)} = (3.008, 1.999)^T$  with a function value equal to 0.018.

We continue this process until one of the termination criteria is satisfied. The progress of the algorithm is shown in Figure 3.12. At the end of this chapter, we present a FORTRAN code implementing the steepest descent algorithm. The code uses the bounding phase and the golden section search methods for performing the unidirectional search.

### 3.4.2 Newton's Method

Newton's method uses second-order derivatives to create search directions. This allows faster convergence to the minimum point. Considering the first three terms in Taylor's series expansion of a multivariable function, it can be shown that the first-order optimality condition will be satisfied if a search direction

$$s^{(k)} = -[\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)})$$

is used. It can also be shown that if the matrix  $[\nabla^2 f(x^{(k)})]^{-1}$  is positive-semidefinite, the direction  $s^{(k)}$  must be descent. But if the matrix  $[\nabla^2 f(x^{(k)})]^{-1}$  is not positive-semidefinite, the direction  $s^{(k)}$  may or may not be descent, depending on whether the quantity  $\nabla f(x^{(k)})^T [\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)})$  is positive or not. Thus, the above search direction may not always guarantee a decrease in the function value in the vicinity of the current point. But the second-order optimality condition suggests that  $\nabla^2 f(x^*)$  be positive-definite (Section 3.1) for the minimum point. Thus, it can be assumed that the matrix  $\nabla^2 f(x^*)$  is positive-definite in the vicinity of the minimum point and the above search direction becomes descent near the minimum point. Thus, this

---

<sup>5</sup>Ideally, the bounding phase method should be used to bracket the minimum. Thereafter, the golden section search method may be used within the obtained interval.

method is suitable and efficient when the initial point is close to the optimum point. Since the function value is not guaranteed to reduce at every iteration, occasional restart of the algorithm from a different point is often necessary.

### Algorithm

The algorithm is the same as Cauchy's method except that Step 4 is modified as follows:

**Step 4** Perform a unidirectional search to find  $\alpha^{(k)}$  using  $\epsilon_2$  such that  $f(x^{(k+1)}) = f(x^{(k)} - \alpha^{(k)} [\nabla^2 f(x^{(k)})]^{-1} \nabla f(x^{(k)})$  is minimum.

### EXERCISE 3.4.3

We choose the Himmelblau function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** As in the previous exercise problem, we choose  $M = 100$ , and initial point  $x^{(0)} = (0, 0)^T$  and termination parameters equal to  $10^{-3}$ . We also set an iteration counter  $k = 0$ .

**Step 2** The derivative at this point, as calculated using Equation (3.7), is  $(-14, -22)^T$ .

**Step 3** Since the termination criteria are not met, we proceed to Step 4.

**Step 4** The search direction involves calculation of the Hessian matrix,  $\nabla^2 f(x^{(0)})$ . We compute second-order partial derivatives

$$\frac{\partial^2 f(x)}{\partial x_1^2}, \quad \frac{\partial^2 f(x)}{\partial x_1 \partial x_2}, \quad \frac{\partial^2 f(x)}{\partial x_2^2}$$

using Equations (3.8) and (3.9). At the point  $x^{(0)}$ , the Hessian matrix and the corresponding inverse are given below:

$$\nabla^2 f(x^{(0)}) = \begin{pmatrix} -42.114 & 0 \\ 0 & -25.940 \end{pmatrix},$$

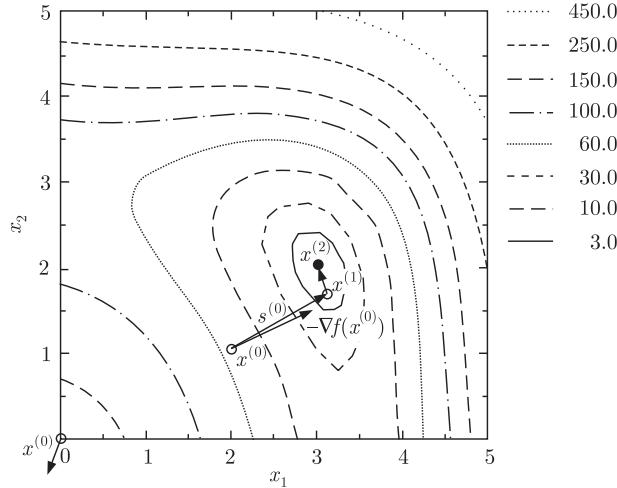
$$(\nabla^2 f(x^{(0)}))^{-1} = \frac{1}{1092.4} \begin{pmatrix} -25.940 & 0 \\ 0 & -42.114 \end{pmatrix}.$$

Since one of the two principal determinants of the above matrix is negative, the matrix is not positive-semidefinite. Thus, we are not sure whether the search along the direction  $s^{(0)}$  will improve the objective function value or not. To be sure, we may compute the quantity  $\nabla f(x^{(0)})^T [\nabla^2 f(x^{(0)})]^{-1} \nabla f(x^{(0)})$  and find its value to be  $-23.274$ , which is a negative quantity. Thus, we can conclude that the search direction  $s^{(0)}$  is non-descent. Nevertheless, we carry on with the steps of the algorithm and investigate the descent property of the search direction.

Using the derivative and inverse Hessian matrix, we can compute the search direction. Along that direction, we specify any point as follows:

$$\begin{aligned} x^{(1)} &= x^{(0)} - \alpha^{(0)} [\nabla^2 f(x^{(0)})]^{-1} \nabla f(x^{(0)}), \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} - \alpha^{(0)} \frac{1}{1092.4} \begin{pmatrix} -25.940 & 0 \\ 0 & -42.114 \end{pmatrix} \begin{pmatrix} -14 \\ -22 \end{pmatrix}, \\ &= \begin{pmatrix} -0.333\alpha^{(0)} \\ -0.846\alpha^{(0)} \end{pmatrix}. \end{aligned}$$

Performing a unidirectional search along this direction, we obtain  $\alpha^{(0)*} = -3.349$ . Since this quantity is negative, the function value does not reduce in the given search direction. Instead, the function value reduces in the opposite direction (refer to Figure 3.13). This demonstrates that the search direction



**Figure 3.13** Two iterations of Newton's method. When the initial point  $(0, 0)^T$  is chosen, the search direction is not descent.

in Newton's method may not always be descent. When this happens, the algorithm is usually restarted with a new point. Thus, we try with another initial point.

**Step 1** We choose an initial point  $x^{(0)} = (2, 1)^T$ . The function value at this point is  $f(x^{(0)}) = 52$ .

**Step 2** The derivative at this point is  $\nabla f(x^{(0)}) = (-56, -28)^T$ .

**Step 3** Since the termination criteria are not met at this point, we move to Step 4.

**Step 4** At this point, the Hessian matrix is given as follows:

$$\nabla^2 f(x^{(0)}) = \begin{pmatrix} 9.994 & 11.940 \\ 11.940 & -6.027 \end{pmatrix}.$$

Calculating the inverse of this matrix, we write a generic point along the new search direction as follows:

$$\begin{aligned} x^{(1)} &= x^{(0)} - \alpha^{(0)} [\nabla^2 f(x^{(0)})]^{-1} \nabla f(x^{(0)}), \\ &= \begin{pmatrix} 2 \\ 1 \end{pmatrix} - \alpha^{(0)} \frac{1}{-202.8} \begin{pmatrix} -6.027 & -11.940 \\ -11.940 & 9.994 \end{pmatrix} \begin{pmatrix} -56 \\ -28 \end{pmatrix}, \\ &= \begin{pmatrix} 2 + 3.313\alpha^{(0)} \\ 1 + 1.917\alpha^{(0)} \end{pmatrix}. \end{aligned}$$

A unidirectional search along this direction reveals that the minimum occurs at  $\alpha^{(0)*} = 0.342$ . Since this quantity is positive, we accept this search direction and calculate the new point:  $x^{(1)} = (3.134, 1.656)^T$ . The function value at this point is 1.491, which is substantially smaller than the initial function value. The search direction  $s^{(0)}$  and the resulting point  $x^{(1)}$  are shown in Figure 3.13.

**Step 5** Assuming that we do not terminate the algorithm at this iteration, we increment the iteration counter to  $k = 1$  and proceed to Step 2. This completes one iteration of Newton's method.

**Step 2** The derivative vector (computed numerically) at this point is  $\nabla f(x^{(1)}) = (3.757, -6.485)^T$ .

**Step 3** The magnitude of the derivative vector is not small. Thus, we move to Step 4 to find a new search direction.

**Step 4** The Hessian matrix at this point is computed numerically using Equations (3.8) and (3.9). Any generic point along the search direction is calculated as follows:

$$\begin{aligned} x^{(2)} &= x^{(1)} - \alpha^{(1)} [\nabla^2 f(x^{(1)})]^{-1} \nabla f(x^{(1)}), \\ &= \begin{pmatrix} 3.134 \\ 1.656 \end{pmatrix} - \alpha^{(1)} \frac{1}{1236.6} \begin{pmatrix} 19.441 & -19.158 \\ -19.158 & 82.488 \end{pmatrix} \begin{pmatrix} 3.757 \\ -6.485 \end{pmatrix}, \\ &= \begin{pmatrix} 3.134 - 0.160\alpha^{(1)} \\ 1.656 + 0.491\alpha^{(1)} \end{pmatrix}. \end{aligned}$$

The minimum point is  $\alpha^{(1)*} = 0.7072$  and the corresponding point is  $x^{(2)} = (3.021, 2.003)^T$  with a function value  $f(x^{(2)}) = 0.0178$ .

We continue to compute one more iteration of this algorithm and find the point  $x^* = (3.000, 2.000)^T$  (up to three decimal places of accuracy) with the function value  $f(x^{(3)}) = 0$ . As discussed earlier, this method is effective for initial points close to the optimum. Therefore, this demands some knowledge of the optimum point in the search space. Moreover, the computations of the Hessian matrix and its inverse are also computationally expensive.

### 3.4.3 Marquardt's Method

Cauchy's method works well when the initial point is far away from the minimum point and Newton's method works well when the initial point is near the minimum point. In any given problem, it is usually not known whether the chosen initial point is away from the minimum or close to the minimum, but wherever be the minimum point, a method can be devised to take advantage of both these methods. In Marquardt's method, Cauchy's method is initially followed. Thereafter, Newton's method is adopted. The transition from Cauchy's method to Newton's method is adaptive and depends on the history of the obtained intermediate solutions, as outlined in the following algorithm.

#### Algorithm

**Step 1** Choose a starting point,  $x^{(0)}$ , the maximum number of iterations,  $M$ , and a termination parameter,  $\epsilon$ . Set  $k = 0$  and  $\lambda^{(0)} = 10^4$  (a large number).

**Step 2** Calculate  $\nabla f(x^{(k)})$ .

**Step 3** If  $\|\nabla f(x^{(k)})\| \leq \epsilon$  or  $k \geq M$ ? **Terminate**;  
Else go to Step 4.

**Step 4** Calculate  $s(x^{(k)}) = -[H^{(k)} + \lambda^{(k)}I]^{-1}\nabla f(x^{(k)})$ . Set  $x^{(k+1)} = x^{(k)} + s(x^{(k)})$ .

**Step 5** Is  $f(x^{(k+1)}) < f(x^{(k)})$ ? If yes, go to Step 6;  
Else go to Step 7.

**Step 6** Set  $\lambda^{(k+1)} = \frac{1}{2}\lambda^{(k)}$ ,  $k = k + 1$ , and go to Step 2.

**Step 7** Set  $\lambda^{(k)} = 2\lambda^{(k)}$  and go to Step 4.

A large value of the parameter  $\lambda$  is used initially. Thus, the Hessian matrix has little effect on the determination of the search direction (see Step 4 of the above algorithm). Initially, the search is similar to that in Cauchy's method. After a number of iterations (when hopefully the current solution has converged close to the minimum), the value of  $\lambda$  becomes small and the effect is more like that in Newton's method. The algorithm can be made faster by performing a unidirectional search while finding the new point in Step 4:

$x^{(k+1)} = x^{(k)} + \alpha^{(k)} s(x^{(k)})$ . Since the computations of the Hessian matrix and its inverse are computationally expensive, the unidirectional search along  $s^{(k)}$  is not usually performed. For simpler objective functions, however, a unidirectional search in Step 4 can be achieved to find the new point  $x^{(k+1)}$ .

### EXERCISE 3.4.4

We consider once again the Himmelblau function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** We begin with a point  $x^{(0)} = (0, 0)^T$ ,  $M = 100$ , and termination factor  $10^{-3}$ . We set a counter  $k = 0$  and a parameter  $\lambda^{(0)} = 100$ . Intentionally, a smaller value of  $\alpha^{(0)}$  is chosen to show the progress of the algorithm in a few iterations. But in practice, a larger value must be chosen to ensure a smooth transition from Cauchy's to Newton's method.

**Step 2** The derivative at this point is  $\nabla f(x^{(0)}) = (-14, -22)^T$ .

**Step 3** Since the derivative is not small and  $k = 0$ , we move to Step 4.

**Step 4** In order to calculate the search direction, we need to compute the Hessian matrix. We have already computed the Hessian matrix at the point  $(0, 0)^T$  in the previous exercise. Using that result, we compute the search direction as follows:

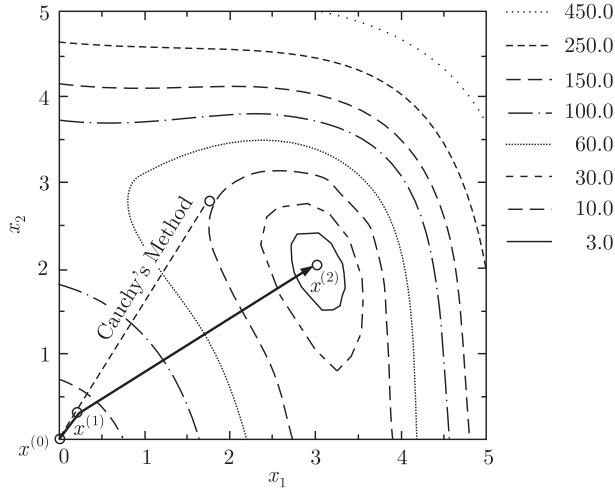
$$\begin{aligned} s^{(0)} &= -[H^{(0)} + \lambda^{(0)} I]^{-1} \nabla f(x^{(0)}), \\ &= -\left[\begin{pmatrix} -42.114 & 0 \\ 0 & -25.940 \end{pmatrix} + 100 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right]^{-1} \begin{pmatrix} -14 \\ -22 \end{pmatrix}, \\ &= -\left[\begin{pmatrix} 57.886 & 0 \\ 0 & 74.060 \end{pmatrix}\right]^{-1} \begin{pmatrix} -14 \\ -22 \end{pmatrix}, \\ &= \begin{pmatrix} 0.242 \\ 0.297 \end{pmatrix}. \end{aligned}$$

Thus, the new point is  $x^{(1)} = x^{(0)} + s^{(0)} = (0.242, 0.297)^T$ .

**Step 5** The function value at this point is  $f(x^{(1)}) = 157.79$ , which is smaller than that at  $x^{(0)}$ , (recall that  $f(x^{(0)}) = 170$ ). Thus, we move to Step 6.

**Step 6** We set a new  $\lambda^{(1)} = 100/2 = 50$ . This has an effect of switching from Cauchy's to Newton's method. We set  $k = 1$  and go to Step 2. This completes one iteration of Marquardt's algorithm. The point  $x^{(1)}$  is shown in Figure 3.14.

**Step 2** The derivative at this point is  $\nabla f(x^{(1)}) = (-23.645, -29.213)^T$ .



**Figure 3.14** Three iterations of Marquardt's method. The deviation from the Cauchy's method is highlighted.

**Step 3** Since the termination criteria are not met, we move to Step 4.

**Step 4** At this step, a new search direction is calculated by computing the Hessian matrix at the current point  $x^{(1)}$ :  $s^{(1)} = (2.738, 1.749)^T$ . Thus, the next point is

$$x^{(2)} = x^{(1)} + s^{(1)} = (2.980, 2.046)^T$$

with a function value  $f(x^{(2)}) = 0.033$ .

**Step 5** Since this point is found to be better than  $x^{(1)}$ , we decrease the parameter  $\lambda$  further:  $\lambda^{(2)} = 25$ .

The process continues until one of the termination criteria is satisfied. After one more iteration, the new point is found to be  $x^{(3)} = (2.994, 2.005)^T$ , having a function value equal to 0.001. Figure 3.14 shows how Marquardt's algorithm converges the search near the optimum point. The difference between Cauchy's method and Marquardt's method is also shown in the same figure. One difficulty with Marquardt's method is the need for computing the Hessian matrix at every iteration. In the following subsections, we present two algorithms that use only the first-order derivatives to find a search direction.

#### 3.4.4 Conjugate Gradient Method

The conjugate gradient method is similar to the conjugate direction method, as discussed in Section 3.3.4. Assuming that the objective function is quadratic (which is a valid assumption at the vicinity of the minimum for many functions), conjugate search directions can be found using only the first-order derivatives. The details of the calculation procedure are beyond the

scope of this book. Interested readers may refer to Reklaitis et al. (1983) or Rao (1984). Fletcher and Reeves (1964) suggested the following conjugate search directions and proved that  $s^{(k)}$  is conjugate to all previous search directions  $s^{(i)}$  for  $i = 1, 2, \dots, (k-1)$ :

$$s^{(k)} = -\nabla f(x^{(k)}) + \frac{\|\nabla f(x^{(k)})\|^2}{\|\nabla f(x^{(k-1)})\|^2} s^{(k-1)}, \quad (3.11)$$

with  $s^{(0)} = -\nabla f(x^{(0)})$ . Note that this recursive equation for search direction  $s^{(k)}$  requires only first-order derivatives at two points  $x^{(k)}$  and  $x^{(k-1)}$ . The initial search direction  $s^{(0)}$  is assumed to be the steepest descent direction at the initial point. Thereafter, the subsequent search directions are found by using the above recursive equation.

### Algorithm

**Step 1** Choose  $x^{(0)}$  and termination parameters  $\epsilon_1, \epsilon_2, \epsilon_3$ .

**Step 2** Find  $\nabla f(x^{(0)})$  and set  $s^{(0)} = -\nabla f(x^{(0)})$ .

**Step 3** Find  $\lambda^{(0)}$  such that  $f(x^{(0)} + \lambda^{(0)} s^{(0)})$  is minimum with termination parameter  $\epsilon_1$ . Set  $x^{(1)} = x^{(0)} + \lambda^{(0)*} s^{(0)}$  and  $k = 1$ . Calculate  $\nabla f(x^{(1)})$ .

**Step 4** Set

$$s^{(k)} = -\nabla f(x^{(k)}) + \frac{\|\nabla f(x^{(k)})\|^2}{\|\nabla f(x^{(k-1)})\|^2} s^{(k-1)}.$$

**Step 5** Find  $\lambda^{(k)}$  such that  $f(x^{(k)} + \lambda^{(k)} s^{(k)})$  is minimum with termination parameter  $\epsilon_1$ . Set  $x^{(k+1)} = x^{(k)} + \lambda^{(k)*} s^{(k)}$ .

**Step 6** Is  $\frac{\|(x^{(k+1)} - x^{(k)})\|}{\|x^{(k)}\|} \leq \epsilon_2$  or  $\|\nabla f(x^{(k+1)})\| \leq \epsilon_3$ ?

If yes, **Terminate**;

Else set  $k = k + 1$  and go to Step 4.

For minimization of linear or quadratic objective functions, two iterations of this algorithm are sufficient to find the minimum. But, for other functions, more iterations through Steps 4 to 6 may be necessary. It is observed that the search directions obtained using Equation (3.11) become linearly dependent after a few iterations of the above algorithm. When this happens, the search process becomes slow. In order to make the search faster, the linear dependence of the search directions may be checked at every iteration. One way to compute the extent of linear dependence is to calculate the included angle between two consecutive search directions  $s^{(k-1)}$  and  $s^{(k)}$ . If the included angle is close to zero (less than a small predefined threshold angle), the algorithm is restarted from Step 1 with  $x^{(0)}$  being the current point. A restart is usually necessary after every  $N$  search directions are created.

**EXERCISE 3.4.5**

Consider the Himmelblau function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** Let us choose the initial point to be  $x^{(0)} = (0, 0)^T$ . All termination parameters are chosen as  $10^{-3}$ .

**Step 2** The derivative vector at  $x^{(0)}$ , calculated numerically, is equal to  $(-14, -22)^T$ . Thus, the initial search direction is  $s^{(0)} = -\nabla f(x^{(0)}) = (14, 22)^T$ .

**Step 3** A unidirectional search from  $x^{(0)}$  along  $s^{(0)}$  using the golden section search finds the minimum point  $x^{(1)} = (1.788, 2.810)^T$ . We set the iteration counter,  $k = 1$ . The derivative vector at the new point is  $\nabla f(x^{(1)}) = (-30.707, 18.803)^T$ . Once the new point is found, we have to create a new search direction conjugate to  $s^{(0)}$ . We find that search direction in the next step.

**Step 4** A new search direction  $s^{(1)}$  is calculated using Equation (3.11):

$$\begin{aligned} s^{(1)} &= -\nabla f(x^{(1)}) + \frac{\|\nabla f(x^{(1)})\|^2}{\|\nabla f(x^{(0)})\|^2} s^{(0)}, \\ &= -\begin{pmatrix} -30.707 \\ 18.803 \end{pmatrix} + \left[ \frac{(-30.707)^2 + (18.803)^2}{(-14)^2 + (-22)^2} \right] \begin{pmatrix} 14 \\ 22 \end{pmatrix}, \\ &= \begin{pmatrix} 57.399 \\ 23.142 \end{pmatrix}. \end{aligned}$$

The unit vector along this search direction is  $(0.927, 0.374)^T$ . In order to check the linear independence of the search directions  $s^{(0)}$  and  $s^{(1)}$ , let us compute the angle between them:

$$\begin{aligned} \cos^{-1} \left[ \frac{s^{(0)}}{\|s^{(0)}\|} \cdot \frac{s^{(1)}}{\|s^{(1)}\|} \right] &= \cos^{-1} \left[ (0.537, 0.844) \begin{pmatrix} 0.927 \\ 0.374 \end{pmatrix} \right], \\ &= \cos^{-1} 0.813 = 35.6^\circ, \end{aligned}$$

which is not close to zero. Thus, the two vectors  $s^{(0)}$  and  $s^{(1)}$  are not linearly dependent. This can also be observed in Figure 3.15.

**Step 5** Performing another unidirectional search along  $s^{(1)}$  from  $x^{(1)}$ , we find the minimum  $x^{(2)} = (2.260, 2.988)^T$ .

**Step 6** The quantities

$$(\|x^{(2)} - x^{(1)}\|)/\|x^{(1)}\| = 0.151,$$

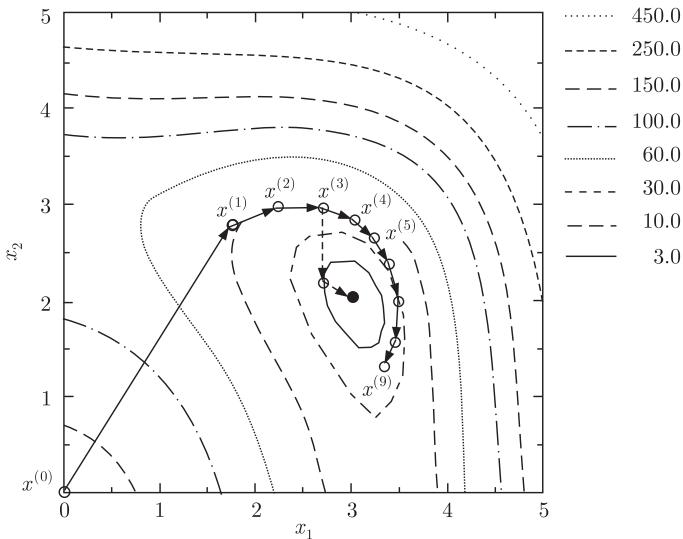
$$\|\nabla f(x^{(2)})\| = \|(-17.875, 44.259)\| = 47.732$$

are both greater than the chosen termination factors. Thus, we increment the iteration counter  $k$  to 2 and move to Step 4. This step completes one iteration of the Fletcher-Reeves method. We observe that in the first iteration, the function value has improved from an initial value of 170.0 to 25.976. Points  $x^{(1)}$  and  $x^{(2)}$  are shown in the contour plot in Figure 3.15.

**Step 4** In the beginning of the second iteration, we first create a search direction  $s^{(2)}$  which should be linearly independent to the direction  $s^{(1)}$ . Using Equation (3.11), we find the new direction:

$$s^{(2)} = -\begin{pmatrix} -17.875 \\ 44.259 \end{pmatrix} + 1.757 \begin{pmatrix} 57.399 \\ 23.142 \end{pmatrix} = \begin{pmatrix} 118.746 \\ -3.590 \end{pmatrix}.$$

We observe that the angle between  $s^{(1)}$  and  $s^{(2)}$  is  $23.7^\circ$ . The new direction is shown in Figure 3.15.



**Figure 3.15** Several iterations of the conjugate gradient method. The convergence to the minimum is slow after a few iterations because of the linear dependence of the search directions.

**Step 5** In order to find the best point along  $s^{(2)}$ , we perform another unidirectional search and find the new point  $x^{(3)} = (2.708, 2.974)^T$  with a function value  $f(x^{(3)}) = 21.207$ . Note that the point  $x^{(3)}$  is better than the point  $x^{(2)}$ .

The second iteration will be complete after checking the termination criteria. In the third iteration, another search direction  $s^{(3)}$  is computed and the angle between directions  $s^{(2)}$  and  $s^{(3)}$  is found to be  $20^\circ$ . A unidirectional search finds a new point  $x^{(4)} = (3.014, 2.852)^T$  with a function value

$f(x^{(4)}) = 18.082$ . The algorithm continues this way until the termination criteria are met. We continue simulating the algorithm for another five iterations and find the point  $x^{(9)} = (3.317, 1.312)^T$  having a function value equal to 5.576.

As seen from Figure 3.15, after a few iterations, search directions tend to become less independent from each other. This causes many iterations of the above algorithm to converge to the optimum solution. If we restart the algorithm at the point  $x^{(3)}$ , then we reassign  $x^{(0)} = (2.708, 2.974)^T$  and move to Step 2. The search direction at this point is  $s^{(0)} = -\nabla f(x^{(0)}) = (-1.610, -52.784)^T$ . A unidirectional search along this direction finds the new point  $(2.684, 2.179)^T$ . After one more search, we obtain the point  $(2.992, 2.072)^T$  with a function value equal to 0.082, which is much better than even the point  $x^{(9)}$  observed without restart. Figure 3.15 also shows how this restart at the point  $x^{(3)}$  improves the search process (with dashed lines).

### 3.4.5 Variable-metric Method (DFP Method)

We have discussed earlier that once the current solution is close to the minimum point of a function, Newton's method is very efficient. But one difficulty with that method is the computation of the inverse of the Hessian matrix. In variable-metric methods, an estimate of the inverse of the Hessian matrix at the minimum point is obtained by iteratively using first-order derivatives. This eliminates expensive computation of the Hessian matrix and its inverse. We replace the inverse of the Hessian matrix by a matrix  $A^{(k)}$  at iteration  $k$  and have a search direction

$$s^{(k)} = -A^{(k)} \nabla f(x^{(k)}). \quad (3.12)$$

Starting with an identity matrix,  $A^{(0)} = I$ , we modify the matrix  $A$  at every iteration to finally take the shape of the inverse of the Hessian matrix at the minimum point. Davidon, Fletcher, and Powell have suggested a way to modify the matrix  $A$  (Davidon, 1959; Fletcher and Powell, 1963). That method is commonly known as the DFP method:

$$\begin{aligned} A^{(k)} &= A^{(k-1)} + \frac{\Delta x^{(k-1)} \Delta x^{(k-1)T}}{\Delta x^{(k-1)T} \Delta e(x^{(k-1)})} \\ &\quad - \frac{A^{(k-1)} \Delta e(x^{(k-1)}) (A^{(k-1)} \Delta e(x^{(k-1)}))^T}{\Delta e(x^{(k-1)})^T A^{(k-1)} \Delta e(x^{(k-1)})}. \end{aligned} \quad (3.13)$$

In the above equation, the quantity  $e(x^{(k)})$  is the gradient of the function at point  $x^{(k)}$ , or  $e(x^{(k)}) = \nabla f(x^{(k)})$ . The change in the design variable vector is denoted by

$$\Delta x^{(k-1)} = x^{(k)} - x^{(k-1)}$$

and the change in gradient vector is denoted by

$$\Delta e(x^{(k-1)}) = e(x^{(k)}) - e(x^{(k-1)}).$$

In the DFP method, the modification of the matrix, using Equation (3.13) preserves the symmetry and the positive-definiteness of the matrix. This property makes the DFP method attractive. Let us recall that in order to achieve a descent direction with Newton's method, the Hessian matrix must be positive-semidefinite. Since an identity matrix is symmetric and positive-definite, at every iteration the positive-definiteness of the matrix  $A^{(k)}$  is retained by the above transformation and the function value is guaranteed to decrease monotonically. Thus, a monotonic improvement in function values in every iteration is expected with the DFP method.

### Algorithm

The algorithm is the same as the Fletcher-Reeves algorithm except that the expression for the search direction  $s(x^{(k)})$  in Step 4 is set according to Equation (3.12) and the matrix  $A^{(k)}$  is calculated using Equation (3.13).

### EXERCISE 3.4.6

Let us consider the Himmelblau function again:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

The inverse of the Hessian matrix at the minimum point  $x^* = (3, 2)^T$  is found to be

$$H^{-1}(x^*) = \begin{pmatrix} 0.016 & -0.009 \\ -0.009 & 0.035 \end{pmatrix}. \quad (3.14)$$

Successive iterations of the DFP method transform an initial identity matrix into the above matrix. This allows the DFP search to become similar to Newton's search after a few iterations. The advantage of this method over Newton's method is that the inverse of the Hessian matrix need not be calculated. Thus, the algorithm has the effect of a second-order search, but the search is achieved only with first-order derivatives.

**Step 1** Once again we begin with the initial point  $x^{(0)} = (0, 0)^T$ . The termination parameters are all set to be  $10^{-3}$ .

**Step 2** The derivative vector at the initial point is equal to  $\nabla f(x^{(0)}) = (-14, -22)^T$ . The search direction is  $s^{(0)} = (14, 22)^T$ .

**Step 3** A unidirectional search from  $x^{(0)}$  along  $s^{(0)}$  gives the minimum point  $x^{(1)} = (1.788, 2.810)^T$ , as found in Step 3 of the Exercise 3.4.5. We calculate the gradient at this point:

$$\nabla f(x^{(1)}) = (-30.707, 18.803)^T.$$

**Step 4** In order to calculate the new search direction, we first compute the parameters required to be used in Equation (3.13):

$$\begin{aligned}\Delta x^{(0)} &= x^{(1)} - x^{(0)} = (1.788, 2.810)^T, \\ \Delta e(x^{(0)}) &= e(x^{(1)}) - e(x^{(0)}) = (-16.707, 40.803)^T, \\ A^{(0)} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.\end{aligned}$$

With these parameters, we compute the next estimate of the matrix  $A$  as follows:

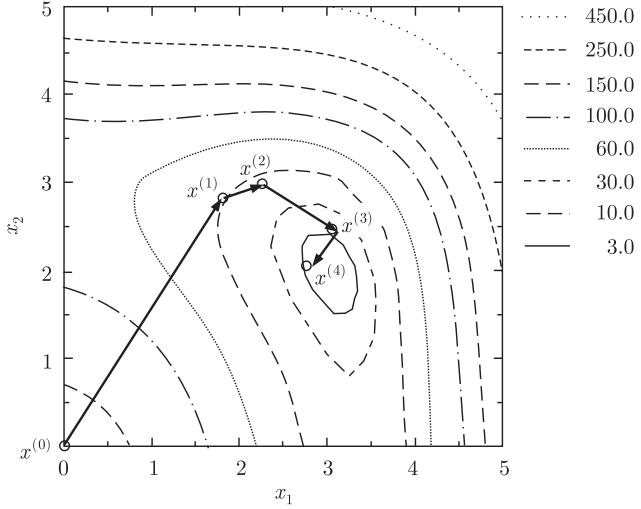
$$\begin{aligned}A^{(1)} &= A^{(0)} + \frac{\Delta x^{(0)} \Delta x^{(0)T}}{\Delta x^{(0)T} \Delta e(x^{(0)})} - \frac{A^{(0)} \Delta e(x^{(0)}) (A^{(0)} \Delta e(x^{(0)}))^T}{\Delta e(x^{(0)})^T A^{(0)} \Delta e(x^{(0)})}, \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \frac{\begin{pmatrix} 1.778 \\ 2.810 \end{pmatrix} (1.788, 2.810)}{(1.788, 2.810) \begin{pmatrix} -16.707 \\ 40.803 \end{pmatrix}} \\ &\quad - \frac{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -16.707 \\ 40.803 \end{pmatrix} \left( \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -16.707 \\ 40.803 \end{pmatrix} \right)^T}{(-16.707, 40.803) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -16.707 \\ 40.803 \end{pmatrix}}, \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 0.038 & 0.060 \\ 0.060 & 0.093 \end{pmatrix} - \begin{pmatrix} 0.144 & -0.351 \\ -0.351 & 0.856 \end{pmatrix}, \\ &= \begin{pmatrix} 0.894 & 0.411 \\ 0.411 & 0.237 \end{pmatrix}.\end{aligned}$$

Knowing  $A^{(1)}$ , we now obtain the search direction  $s^{(1)}$  using Equation (3.12) as follows:

$$s^{(1)} = - \begin{pmatrix} 0.894 & 0.411 \\ 0.411 & 0.237 \end{pmatrix} \begin{pmatrix} -30.707 \\ 18.803 \end{pmatrix} = \begin{pmatrix} 19.724 \\ 8.164 \end{pmatrix}.$$

The unit vector along this direction is  $(0.924, 0.382)^T$ . It is interesting to observe that this search direction is similar to the first search direction  $s^{(1)}$  obtained by using the Fletcher-Reeves algorithm (see Exercise 3.4.5). But soon we shall see how subsequent search directions obtained by the DFP method vary from the Fletcher-Reeves method.

**Step 5** Performing a unidirectional search along  $s^{(1)}$  from the point  $x^{(1)}$ , we obtain the minimum point:  $x^{(2)} = (2.248, 2.989)^T$  having a function value  $f(x^{(2)}) = 26.237$  (Figure 3.16).



**Figure 3.16** Three iterations of the DFP method shown on a contour plot of Himmelblau's function.

**Step 6** We observe that the point  $x^{(2)}$  is very different from the point  $x^{(1)}$  and calculate the gradient at  $x^{(2)}$ :  $\nabla f(x^{(2)}) = (-18.225, 44.037)^T$ . Since  $\|\nabla f(x^{(2)})\|$  is not close to zero, we increment  $k$  and proceed to Step 4. This completes one iteration of the DFP method. The initial point  $x^{(0)}$ , the point  $x^{(1)}$ , and the point  $x^{(2)}$  are shown in Figure 3.16.

**Step 4** The second iteration begins with computing another search direction  $s^{(2)} = -A^{(2)}\nabla f(x^{(2)})$ . The matrix  $A^{(2)}$  can be computed by calculating  $\Delta x^{(1)}$  and  $\Delta e(x^{(1)})$  as before and using Equation (3.13). By simplifying the expression, we obtain the new matrix

$$A^{(2)} = \begin{pmatrix} 0.070 & -0.017 \\ -0.017 & 0.015 \end{pmatrix}$$

and the new search direction is found to be

$$s^{(2)} = (0.731, -0.976)^T.$$

Note that this direction is more descent than that obtained in the Fletcher-Reeves method after one iteration.

**Step 5** The unidirectional search along  $s^{(2)}$  finds the best point:  $x^{(3)} = (2.995, 1.991)^T$  with a function value equal to  $f(x^{(3)}) = 0.003$ .

Another iteration updates the  $A$  matrix as follows:

$$A^{(3)} = \begin{pmatrix} 0.030 & -0.005 \\ -0.005 & 0.020 \end{pmatrix}.$$

The new search direction is found to be  $s^{(3)} = (0.014, 0.095)^T$ . The unidirectional search method finds the new point  $x^{(4)} = (2.997, 2.003)^T$  with a function value equal to  $3 \times 10^{-4}$ . Another iteration finds the following  $A$  matrix:

$$A^{(4)} = \begin{pmatrix} 0.018 & -0.011 \\ -0.011 & 0.036 \end{pmatrix}$$

and the new search direction  $s^{(4)} = (0.003, -0.003)^T$ . A unidirectional search finds the point  $x^{(5)} = (3.000, 2.000)^T$  with a function value equal to zero. One final iteration finds the  $A$  matrix:

$$A^{(5)} = \begin{pmatrix} 0.016 & -0.009 \\ -0.009 & 0.035 \end{pmatrix},$$

which is identical to the inverse of the Hessian matrix of the Himmelblau function at the minimum point (Equation (3.14)). The above calculation shows how the original identity matrix  $A^{(0)}$  has transformed into  $H^{-1}(x^*)$  matrix in successive iterations. From the fourth iteration onwards, the search direction is very similar to that in the Newton's method, because the matrix  $A^{(k)}$  is similar to the inverse of the Hessian matrix. Since, during that iteration the solution is also close to the minimum, the search converges faster to the minimum point. One difficulty with this method is that the matrix  $A$  sometimes becomes ill-conditioned due to numerical derivative computations and due to inaccuracy involved in unidirectional searches.

### 3.5 Summary

Optimization algorithms have been discussed to solve multivariable functions. At first, optimality conditions have been presented. Thereafter, four direct search algorithms have been discussed followed by five different gradient-based search methods.

In multivariable function optimization, the first-order optimality condition requires that all components of the gradient vector be zero. Any point that satisfies this condition is a likely candidate for the minimum point. The second-order optimality condition for minimization requires the Hessian matrix to be positive-definite. Thus, a point is minimum if both the first and the second-order optimality conditions are satisfied.

Among the direct search methods, the evolutionary optimization method compares  $2^N$  (where  $N$  is the number of design variables) function values at each iteration to find the current best point. This algorithm usually requires a large number of function evaluations to find the minimum point. The simplex search method also uses a number of points ( $(N+1)$  of them) at each iteration to find one new point, which is subsequently used to replace the worst point in the simplex. If the size of the simplex is large, this method has a tendency to wander about the minimum. The Hooke-Jeeves pattern search method

works in two phases. In the first phase, an exploratory move is performed to search in the vicinity of the current point. Thereafter, a pattern move is performed in the direction from the previous point to the current point. The storage requirement is usually more in this algorithm and this algorithm has a tendency to degenerate into many exploratory moves. For quadratic objective functions, this method has a proof of convergence. Finally, Powell's conjugate direction method finds  $N$  conjugate search directions and performs a unidirectional search along each direction, each time starting from the best point found in the previous search. This method is the most popular among all direct search methods.

If the derivative information is not directly available, numerical computation of the first and the second-order derivatives can be obtained. All gradient-based methods work on the principle of generating new search directions iteratively and performing a unidirectional search along each direction. The steepest descent direction is a direction opposite to the gradient direction at any point. In Cauchy's method, the steepest descent direction is used as a search direction, thereby ensuring a monotonic reduction in function values in successive iterations. Cauchy's method works well for solutions far away from the true minimum point. The search becomes slow when the solution is closer to the minimum point. In Newton's method (similar to the Newton-Raphson method for single-variable function optimization), the second-order derivative information (called the Hessian matrix) is used to find a search direction. Even though this search direction is not guaranteed to be a descent direction, for points close to the true minimum point, this method is very efficient. Marquardt's method uses a compromise of both these methods by adaptively blending the transition from Cauchy's method to Newton's method in creating search directions. The Fletcher-Reeves method creates search directions that are conjugate with respect to the objective function and has a similar effect as in Powell's conjugate direction method. The DFP method uses the concept of Newton's method, but the Hessian matrix is not directly used. Instead, the Hessian matrix at the minimum point is estimated by successively modifying an initial identity matrix using first-order derivatives. With this background, we are now ready to discuss algorithms for solving constrained, multivariable optimization problems in the next chapter.

## REFERENCES

- Box, G. E. P. (1957). Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, **6**, 639–641.
- Box, G. E. P. and Draper, N. R. (1969). *Evolutionary Operation*. New York: Wiley.
- Davidon, W. C. (1959): Variable metric method for minimization (Technical Report No. ANL-599). *AEC Research Development Report*.

- Fletcher, R. and Powell, M. J. D. (1963). A rapidly convergent descent method for minimization. *Computer Journal*, **6**, 163–168.
- Fletcher, R. and Reeves, C. M. (1964). Function minimization by conjugate gradients. *Computer Journal*, **7**, 149–154.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.
- Kreyszig, E. (1983). *Advanced Engineering Mathematics*. New Delhi: Wiley Eastern.
- Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, **7**, 308–313.
- Rao, S. S. (1984). *Optimization Theory and Applications*. New Delhi: Wiley Eastern.
- Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. (1983). *Engineering Optimization—Methods and Applications*. New York: Wiley.
- Spendley, W., Hext, G. R., and Himsorth, F. R. (1962). Sequential applications of simplex designs in optimization and evolutionary operation. *Technometrics*, **4**, 441–461.
- Strang, G. (1980). *Linear Algebra and Its Applications*. Orlando: Academic Press.

## PROBLEMS

**3-1** Locate and classify the stationary points of the following functions:

- (i)  $f(x_1, x_2) = x_1^2 + 2x_2^2 - 4x_1 - 2x_1x_2$ .
- (ii)  $f(x_1, x_2) = 10(x_2 - x_1^2)^2 + (1 - x_1)^2$ .
- (iii)  $f(x_1, x_2, x_3) = (x_1 + 3x_2 + x_3)^2 + 4(x_1 - x_3 - 2)^2$ .

**3-2** Find the stationary points of the function:

$$f(x_1, x_2) = 2x_1^3 - 3x_1^2 - 6x_1x_2(x_1 - x_2 - 1).$$

- (i) Which of these points are local minima, which are local maxima, and which are neither?
- (ii) How many minima exist along the line joining  $(0, 1)^T$  and  $(1, 2)^T$ ? Which are these points? Explain.

**3-3** Consider the unconstrained function:

$$f(x_1, x_2) = (x_1^2 - x_2)^2 + x_2^2.$$

Perform five iterations of a unidirectional search using the golden section search method along the following search directions:

- (i)  $s = (2, 1)^T$  from the point  $(-5, 5)^T$  up to the point  $(5, 0)^T$ .
- (ii)  $s = (-1, -1)^T$  from the point  $(5, 5)^T$  up to the point  $(-5, -5)^T$ .

**3-4** Given the function:

$$f(x_1, x_2) = 2 + (x_1^2 - x_2)^2 + x_2^2.$$

Using three iterations of the golden section search method, estimate the minimum point along the line joining the points  $(-3, -4)^T$  and  $(3, 2)^T$ . Restrict the search between the above two points.

**3-5** In trying to find the maximum of the following function, the point  $x^{(t)} = (3, 2, -1)^T$  is encountered:

$$f(x_1, x_2, x_3) = 6x_1^2 x_2 - 5x_3 + 2x_1(x_2^2 + x_3^2).$$

Determine whether the search direction  $s^{(t)} = (-1, 1, -2)^T$  would be able to find better solutions locally from  $x^{(t)}$ .

**3-6** Given the function:

$$f(x) = 10 - x_1 + x_1 x_2 + x_2^2,$$

an initial point  $x^{(0)} = (2, 4)^T$ , and a direction vector  $d^{(0)} = (-1, -1)^T$ , such that any point along  $d^{(0)}$  can be written as  $x(\alpha) = x^{(0)} + \alpha d^{(0)}$ . Using three iterations of the interval halving method estimate the value of  $\alpha$  in a one-dimensional search along  $d^{(0)}$  for which  $f(\alpha) = 15$ . Assume  $\alpha_{\min} = 0$  and  $\alpha_{\max} = 3$ . Do not solve the quadratic equation to find the value of  $\alpha$ .

**3-7** In solving the following problem:

$$\text{Minimize } f(x_1, x_2) = 10 + x_1^2 - 5x_1 x_2 + 9x_2^2 + x_2$$

using  $x_1 = (-1, 1)^T$  and  $x_2 = (1, -1)^T$  and a search direction  $d = (1, -1)^T$ , we would like to use the conjugate direction method using the parallel subspace property.

- (i) Find the direction  $s$  which is  $C$ -conjugate to  $d$ . Show that  $s$  is  $C$ -conjugate to  $d$ .
- (ii) Continue to find the minimum solution of the above function. Verify this solution by finding the minimum using the first and second-order optimality conditions.

**3-8** In order to minimize the unconstrained objective function

$$x_1^3 - 2x_1 x_2^2 + 4x_1 + 10$$

a search direction  $(\cos \theta, \sin \theta)^T$  needs to be used at the point  $(1, 2)^T$ . What is the range of  $\theta$  for which the resulting search direction is descent? What is the steepest descent direction?

**3-9** In trying to minimize the function

$$f(x_1, x_2) = 10 - x_1 + x_1 x_2 + x_2^2,$$

use  $x^{(1)} = (0, 2)^T$ ,  $x^{(2)} = (0, 0)^T$  and  $x^{(3)} = (1, 1)^T$  as the initial simplex of three points. Complete two iterations of Nelder and Mead's simplex search algorithm to find the new simplex. Assume  $\beta = 0.5$  and  $\gamma = 2$ .

**3-10** Consider the following function for minimization

$$f(x_1, x_2, x_3) = x_1^2 + 2x_2^2 + 2x_3^2 + 2x_1 x_2 + 2x_2 x_3 - 3x_1$$

and a search direction  $s^{(1)} = (1, -1, -1)^T$ . Using two points  $(1, 0, 1)^T$  and  $(0, 1, 0)^T$ , find a new search direction  $s^{(2)}$  conjugate to  $s^{(1)}$  using parallel subspace property. Show that the search direction  $s^{(2)}$  is conjugate to  $s^{(1)}$  with respect to the above function.

**3-11** Find whether the given direction  $s$  at the point  $x$  is descent for the respective functions:

(i) For  $f(x_1, x_2) = 2x_1^2 + x_2^2 - 2x_1 x_2 + 4$ ,

$$s = (1, 1)^T, \quad x = (2, 3)^T.$$

(ii) For  $f(x_1, x_2) = x_1^4 + x_2^3 - 2x_1^2 x_2^2 + 10x_1/x_2^2$ ,

$$s = (-1, 2)^T, \quad x = (0, 1)^T.$$

(iii) For  $f(x_1, x_2, x_3) = (x_1 + 2x_2 + 3x_3)^2 - 5(x_1^2 - x_3)^2$ ,

$$s = (1, 2, -1)^T, \quad x = (1, 0, 1)^T.$$

(iv) For  $f(x_1, x_2, x_3) = x_1^3 x_2 + 0.2 \exp(x_2^2 - x_3)$ ,

$$s = (-10, -20, 5)^T, \quad x = (1, 2, 6)^T.$$

**3-12** Find out whether an exploratory search used in the Hooke-Jeeves pattern search method at  $x^{(0)} = (0.0, 0.5)^T$  with  $\Delta x_1 = \Delta x_2 = 0.8$  is successful in minimizing the following function:

$$f(x_1, x_2) = 10x_1 - x_2 + 2x_1^2 x_2 + x_1 x_2^2 + x_2^3.$$

**3-13** Consider the four-variable minimization problem:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= (x_1 + 2x_2 - 1)^2 + 5(x_3 - x_4)^2 + (x_2 - 3x_3)^4 \\ &\quad + 10(x_1 - x_4)^4. \end{aligned}$$

Perform two iterations of following algorithms from the point  $x^{(0)} = (2, -1, 0, 1)^T$ .

- (i) Hooke-Jeeves method with  $\Delta = (1, 1, 1, 1)^T$ .
- (ii) Cauchy's method.
- (iii) Fletcher-Reeves method.

**3-14** What are the differences between Cauchy's and Newton's search methods? Determine for what values of  $x_1$  and  $x_2$ , Newton's search is guaranteed to be successful for the following unconstrained minimization problem:

$$f(x_1, x_2) = x_1^3 - 4x_1x_2 + x_2^2.$$

**3-15** The steepest descent method applied to the function

$$f(\mathbf{x}) = 2x_1^2 - 2x_1x_2 + x_2^2 + 2x_1 - 2x_2$$

generates a sequence of solutions  $\{\mathbf{x}^{(t)}\}$  for  $t \geq 1$  (where  $t$  is the iteration number). If  $\mathbf{x}^{(2k+1)} = (0, 1 - \frac{1}{5^k})^T$ , show that  $\mathbf{x}^{(2k+3)} = (0, 1 - \frac{1}{5^{k+1}})^T$ .

Starting from  $x^{(1)} = (0, 0)^T$ , sketch next two points on a contour plot of the function in the  $x_1$ - $x_2$  plane.

**3-16** Consider the following minimization problem:

$$f(x_1, x_2) = 2x_1^2 + x_2^2 + 2x_1x_2 + x_1 - x_2.$$

- (i) Use the Fletcher-Reeves's conjugate gradient method to find the minimizer. Start with the initial solution  $x^{(0)} = (0, 0)^T$ .
- (ii) Show that two conjugate directions  $s^{(0)}$  and  $s^{(1)}$  derived from  $x^{(0)}$  are  $C$ -conjugate to each other.
- (iii) If the points  $x^{(0)}$  and  $d^{(0)} = -\nabla f(x^{(0)})$  are used along with any generic point  $x^{(1)} = (a, b)^T$  (such that  $a = b \neq 0$ ), show that the resulting conjugate direction method  $d^{(1)}$  obtained using the parallel subspace property will be identical to  $s^{(1)}$  obtained by the conjugate gradient method.

**3-17** Prove that two consecutive search directions obtained in the steepest descent search algorithm are mutually orthogonal to each other.

**3-18** Solve the following equations by formulating suitable optimization problems:

- (i)  $2x + y = 5, \quad 3x - 2y = 2$ .
- (ii)  $x^2 - 5xy + y^3 = 2, \quad x + 3y = 6$ .
- (iii)  $z \exp(x) - x^2 = 10y, \quad x^2z = 0.5, \quad x + z = 1$ .

Use the computer program listed in the following to obtain the solutions with two decimal places of accuracy.

**3-19** Starting from the point  $(1, 1)^T$ , perform two iterations of DFP method to find a stationary point of the following function:

$$f(x_1, x_2) = 10 - x_1 + x_1 x_2 + x_2^2.$$

**3-20** G. G. Broyden, R. Fletcher, and D. F. Shanno have suggested the following recursive relationship (largely known as the BFS method) for the matrix  $A$  used in the variable-metric method:

$$A^{(k+1)} = \left[ I - \frac{\Delta x^{(k)} (\Delta e^{(k)})^T}{\Delta x^{(k)T} \Delta e^{(k)}} \right] A^{(k)} \left[ I - \frac{\Delta x^{(k)} (\Delta e^{(k)})^T}{\Delta x^{(k)T} \Delta e^{(k)}} \right]$$

$$+ \frac{\Delta x^{(k)} (\Delta x^{(k)})^T}{\Delta x^{(k)T} \Delta e^{(k)}}.$$

The above parameters are explained in Section 3.4.5. Starting from an identity matrix  $A^{(0)} = I$  and an initial point  $x^{(0)} = (0, 0)^T$ , show that the above update formula makes the matrix  $A$  converge to the inverse of the Hessian matrix at the minimum point of the following minimization problem:

$$f(x_1, x_2) = 2x_1^2 - 4x_1x_2 + 4x_2^2 - 4x_1 + 5.$$

## **COMPUTER PROGRAM**

A computer code implementing Cauchy's steepest descent method is presented here. Note how easily the algorithm given in Section 3.4.1 is coded to generate the computer program. Other algorithms can also be coded similarly. In the following, we first outline the code and then present the results from a simulation run. The objective function is coded in the function `funct`. In the following code, Himmelblau's function is used. For any other function, the function `funct` can be suitably modified.

```
c ****
c *
c *          STEEPEST DESCENT SEARCH
c *
c ****
c Developed by Kalyanmoy Deb
c           Indian Institute of Technology, Kanpur
c All rights reserved. This listing is for personal use.
c %%%%%%
c This routine finds the minimum point of a multi
c variable function using Cauchy's steepest descent
c algorithm. Code your function in subroutine funct
c %%%%%%
implicit real*8 (a-h,o-z)
```

```

dimension x0(10),xstar(10),gr0(10),s0(10),x1(10),
-           xd(10)
write(*,*) 'enter the dimension of design vector'
read(*,*) n
write(*,*) 'enter initial vector'
read(*,*) (x0(i),i=1,n)
write(*,*) 'enter accuracy in steepest descent'
read(*,*) eps
write(*,*) 'enter accuracy in golden section'
read(*,*) epss
write(*,*) 'enter 1 for intermediate results'
read(*,*) iprint
nfun = 0
call steepest(n,eps,epss,x0,xstar,fstar,nfun,ierr,
-               gr0,s0,x1,xd,iprint)
if (ierr .ne. 1) then
  write(*,1) (xstar(i),i=1,n)
1   format(/,2x,'The minimum solution is : (',
-             2(f10.4,',','))')
  write(*,2) fstar
2   format(2x,'Function value: ',1pe13.5)
  write(*,3) nfun
3   format(2x,'Number of function evaluations: ',i8)
else
  write(*,*) 'Terminated due to above error.'
endif
stop
end

subroutine steepest(n,eps,epss,x0,xstar,fstar,
-                   nfun,ierr,grad0,s0,x1,xdummy,ip)
c.....steepest Descent method
c.....n : dimension of design vector
c.....eps : accuracy in steepest-descent search
c.....epss : accuracy in golden section search
c.....x0 : initial design vector
c.....xstar : final design solution (output)
c.....fstar : final objective function value (output)
c.....nfun : number of function evaluations required
c.....ierr : error code, 1 for error
c.....rest all are dummy variables of size n
  implicit real*8 (a-h,o-z)
  dimension x0(n),xstar(n),grad0(n),s0(n),x1(n),
-             xdummy(n)
  maxiter = 10000
  k = 0

```

```

c.....step 2 of the algorithm
2  call fderiv(n,x0,grad0,f0,nfun,xdummy)
   if (ip .eq. 1) then
      write(*,*)'-----',
      write(*,8) k,(x0(i),i=1,n)
8   format(2x,'Iteration: ',i5,/,2x,
   -       'Solution vector: (',2(1pe12.4,',','),')')
      write(*,9) f0,nfun
9   format(2x,'Function value: ',1pe13.4,
   -           ' Function Eval. : ',i7)
      endif
c.....step 3 of the algorithm
   call unitvec(n,grad0,gradmag)
   if ((gradmag .le. eps) .or. (k .ge. maxiter)) then
      do 11 i = 1,n
11    xstar(i) = x0(i)
      call funct(n, xstar, fstar, nfun)
      return
      endif
c.....step 4 of the algorithm
   do 10 i = 1,n
      s0(i) = -1.0 * grad0(i)
10  continue
   call bphase(n,x0,s0,a,b,nfun,xdummy)
   call golden(n,x0,s0,a,b,epss,alfastr,nfun,ierr,
   -           xdummy)
   sum = 0.0
   do 12 i = 1,n
      x1(i) = x0(i) + alfastr * s0(i)
      if (dabs(x0(i)) .gt. eps) then
         sum = sum + dabs(x1(i) - x0(i))/x0(i)
      else
         sum = sum + dabs(x1(i) - x0(i))/eps
      endif
12  continue
c.....step 5 of the algorithm
   if (sum .le. eps) then
      do 14 i = 1,n
         xstar(i) = x1(i)
14  continue
      call funct(n, xstar, fstar, nfun)
      return
   else
      k = k + 1
      do 15 i = 1,n
         x0(i) = x1(i)

```

```

15      continue
      go to 2
      endif
      end

      subroutine golden(n,x,s,a,b,eps,xstar,nfun,ierr,
      -           xdummy)
c.....golden section search algorithm
c.....finds xstar such that f(x+xstar*s) is minimum
c.....x : solution vector
c.....s : direction vector
c.....a,b : lower and upper limits
c.....nfun : function evaluations
c.....ierr : error code, 1 for error
c.....xdummy : dummy variable of size n
      implicit real*8 (a-h,o-z)
      real*8 lw,x(n),s(n),xdummy(n)
c.....step 1 of the algorithm
      xstar = a
      ierr=0
      maxfun = 10000
      aw=0.0
      bw=1.0
      lw=1.0
      k=1
c.....golden number
      gold=(sqrt(5.0)-1.0)/2.0
      w1prev = gold
      w2prev = 1.0-gold
c.....initial function evaluations
      call mapfun(n,x,s,a,b,w1prev,fw1,nfun,xdummy)
      call mapfun(n,x,s,a,b,w2prev,fw2,nfun,xdummy)
      ic=0
c.....step 2 of the algorithm
10    w1 = w1prev
      w2 = w2prev
c.....calculate function value for new points only
      if (ic .eq. 1) then
        fw2 = fw1
        call mapfun(n,x,s,a,b,w1,fw1,nfun,xdummy)
      else if (ic .eq. 2) then
        fw1 = fw2
        call mapfun(n,x,s,a,b,w2,fw2,nfun,xdummy)
      else if (ic .eq. 3) then
        call mapfun(n,x,s,a,b,w1,fw1,nfun,xdummy)
        call mapfun(n,x,s,a,b,w2,fw2,nfun,xdummy)

```

```

        endif
c.....region-elimination rule
if (fw1 .lt. fw2) then
  ic = 1
  aw = w2
  lw = bw-aw
  w1prev = aw + gold * lw
  w2prev = w1
else if (fw2 .lt. fw1) then
  ic = 2
  bw = w1
  lw=bw-aw
  w1prev = w2
  w2prev = bw - gold * lw
else
  ic = 3
  aw = w2
  bw = w1
  lw = bw-aw
  w1prev = aw + gold * lw
  w2prev = bw - gold * lw
endif
k=k+1
c.....step 3 of the algorithm
if (dabs(lw) .lt. eps) then
  xstar = a + (b-a) * (aw+bw)/2
  return
else if (nfun .gt. maxfun) then
  write(*,3) maxfun,a+aw*(b-a),a+bw*(b-a)
3   format('Golden section did not converge in',i6,
         -           ' function evaluations',//,'Interval (',
         -           1pe12.5,',',1pe12.5,')')
  ierr = 1
  return
endif
go to 10
end

subroutine bphase(n,x,s,a,b,nfun,xdummy)
c.....bounding phase method
c.....arguments are explained in subroutine golden
  implicit real*8 (a-h,o-z)
  dimension x(n),s(n),xdummy(n)
c.....step 1 of the algorithm
c.....initial guess, change if you like
  w0 = 0.0

```

```

    delta = 1.0
1   call mapfun(n,x,s,0d0,1d0,w0-delta,fn,nfun,xdummy)
    call mapfun(n,x,s,0d0,1d0,w0,f0,nfun,xdummy)
    call mapfun(n,x,s,0d0,1d0,w0+delta,fp,nfun,xdummy)
c.....step 2 of the algorithm
    if (fn .ge. f0) then
        if (f0 .ge. fp) then
            delta = 1 * delta
        else
            a = w0 - delta
            b = w0 + delta
        endif
    elseif ((fn .le. f0) .and. (f0 .le. fp)) then
        delta = -1 * delta
    else
        delta = delta / 2.0
        go to 1
    endif
    k=0
    wn = w0 - delta
c.....step 3 of the algorithm
3   w1 = w0 + (2**k) * delta
    call mapfun(n,x,s,0d0,1d0,w1,f1,nfun,xdummy)
c.....step 4 of the algorithm
    if (f1 .lt. f0) then
c.....bracketing isn't over, reset wn to w0 and w0 to w1
        k = k+1
        wn = w0
        fn = f0
        w0 = w1
        f0 = f1
        go to 3
    else
c.....bracketing is complete, so quit
        a = wn
        b = w1
    endif
    if (b .lt. a) then
        temp = a
        a = b
        b = temp
    endif
    return
end

```

```

subroutine fderiv(n,x,grad,f,nfun,xd)
c.....derivative calculation at point x
implicit real*8 (a-h,o-z)
c.....calculates the first derivative of the function
dimension x(n),grad(n),xd(n)
do 10 i = 1,n
    xd(i) = x(i)
10 continue
call funct(n,xd,f,nfun)
c.....set delta_x
do 12 i = 1,n
    if (xd(i) .lt. 0.01) then
        dx = 0.01
    else
        dx = 0.01 * xd(i)
    endif
    xd(i) = xd(i) + dx
c..... compute two function evaluations
call funct(n,xd,fp,nfun)
xd(i) = xd(i) - 2 * dx
call funct(n,xd,fn,nfun)
c..... then compute the gradient
grad(i) = (fp-fn)/(2.0*dx)
xd(i)=x(i)
12 continue
return
end

subroutine mapfun(n,x,s,a,b,w,f,nfun,xd)
c.....calculates a point and function value in mapped
c.....w units away from a point x in s direction
implicit real*8 (a-h,o-z)
dimension x(n),s(n),xd(n)
xw = a + w * (b-a)
do 10 i = 1,n
    xd(i) = x(i) + xw * s(i)
10 continue
call funct(n,xd,f,nfun)
return
end

subroutine funct(n,x,f,nfun)
c.....calculates the function value at x
implicit real*8 (a-h,o-z)
dimension x(n)
nfun = nfun + 1

```

```

f=(x(1)*x(1)+x(2)-11.0)**2+(x(1)+x(2)*x(2)-7.0)**2
return
end

subroutine unitvec(n,x,sum)
c.....computes a unit vector
implicit real*8 (a-h,o-z)
dimension x(n)
sum = 0.0
do 1 i = 1,n
    sum = sum + x(i)*x(i)
1 continue
sum = dsqrt(sum)
if (sum .le. 1.0e-06) then
    do 2 i = 1,n
        x(i) = x(i)/sum
2 continue
endif
return
end

```

## Simulation Run

The above code is run on a PC-386 using Microsoft FORTRAN compiler for minimizing Himmelblau's function starting from  $x^{(0)} = (0, 0)^T$ . The solution is achieved with three decimal places of accuracy. First, the input to the code is given. Thereafter, the output from the code is presented.

```

enter the dimension of design vector
2
enter initial vector
0 0
enter accuracy in steepest descent
0.001
enter accuracy in golden section
0.001
enter 1 for intermediate results
1
-----
Iteration:      0
Solution vector: (  0.0000E+00,  0.0000E+00,)
Function value:  1.7000E+02 Function Eval. :      5
-----
Iteration:      1
Solution vector: (  1.7880E+00,  2.8098E+00,)

```

```
Function value: 3.2129E+01 Function Eval. : 30
-----
Iteration: 2
Solution vector: ( 3.0079E+00, 1.9987E+00,)
Function value: 2.1108E-03 Function Eval. : 55
-----
Iteration: 3
Solution vector: ( 3.0008E+00, 1.9973E+00,)
Function value: 1.0788E-04 Function Eval. : 80
-----
Iteration: 4
Solution vector: ( 3.0003E+00, 1.9999E+00,)
Function value: 2.2268E-06 Function Eval. : 105
The minimum solution is : ( 3.0000, 1.9999,)
Function value: 1.53374E-07
Number of function evaluations: 126
```

# 4

## Constrained Optimization Algorithms

---

In this chapter, constrained optimization algorithms are discussed. Almost all engineering design and decision making problems have an objective of minimizing or maximizing a function (usually cost or profit functions) and simultaneously have a requirement for satisfying some constraints arising due to space, strength, or stability considerations. The constrained optimization algorithms can be grouped into direct and gradient-based methods. Some of these algorithms employ single-variable and multivariable unconstrained optimization algorithms described in Chapters 2 and 3.

A constrained optimization problem comprises an objective function together with a number of equality and inequality constraints. Often lower and upper bounds on design or decision variables are also specified. Considering that there are  $N$  decision or design variables, we write a constrained problem as follows:

$$\begin{aligned} & \text{Minimize} && f(x) \\ & \text{subject to} && \\ & g_j(x) \geq 0, && j = 1, 2, \dots, J; \\ & h_k(x) = 0, && k = 1, 2, \dots, K; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, && i = 1, 2, \dots, N. \end{aligned} \quad \left. \right\} \quad (4.1)$$

This is the most general form of a single-objective constrained optimization problem. The function  $f(x)$  is the objective function, which is to be minimized. The functions  $g_j(x)$  and  $h_k(x)$  are inequality and equality constraint functions, respectively. There are  $J$  inequality constraints and  $K$  equality constraints in the above problem. In the rest of this book, inequality constraints are

expressed in greater-than-or-equal type. We recognize here that a less-than-or-equal type constraint can be transformed into the other type by multiplying the constraint function by  $-1$ . Since none of the above functions (objective function or constraints) are assumed to be linear, the above problem is also known as a *Nonlinear Programming Problem* or simply an NLP problem. Even though the above problem is written for minimization, maximization problems can also be solved by using the duality principle discussed in Chapter 1.

Any point  $x^{(t)}$  is said to have *satisfied* a constraint, if the left side expression of the constraint at that point agrees with the right-side value by the relational operator between them. A point (or solution) is defined as a *feasible* point (or solution) if all equality and inequality constraints and variable bounds are satisfied at that point. All other points are known as *infeasible* points. It is important to note that an infeasible point can never be the optimum point. We shall discuss the necessary and sufficient conditions for a point to be optimum in the next section. There are two ways an inequality constraint can be satisfied at a point—the point falls either on the constraint surface (that is,  $g_j(x^{(t)}) = 0$ ) or on the feasible side of the constraint surface (where  $g_j(x^{(t)})$  is positive). If the point falls on the constraint surface, the constraint is said to be an *active* constraint at that point. In the latter case, the constraint is *inactive* at the point.

In the remainder of this chapter, we discuss the optimality criteria for constrained optimization problems followed by a number of constrained search algorithms.

## 4.1 Kuhn-Tucker Conditions

In constrained optimization problems, points satisfying Kuhn-Tucker conditions are likely candidates for the optimum. Kuhn-Tucker conditions are described a little later. But it is noteworthy that as in the case of unconstrained optimization problems, not all stationary points are optimal points and in constrained optimization problems, not all Kuhn-Tucker points (points that satisfy Kuhn-Tucker conditions) are optimal points. Nevertheless, many constrained optimization algorithms are designed to find Kuhn-Tucker points. We consider the NLP problem stated in Equation (4.1). Using Lagrange multiplier technique, the inequality and equality constraints can be added to the objective function to form an unconstrained problem. The following Kuhn-Tucker conditions can be obtained by satisfying the first-order optimality conditions of that unconstrained problem (Rao, 1984):

$$\nabla f(x) - \sum_{j=1}^J u_j \nabla g_j(x) - \sum_{k=1}^K v_k \nabla h_k(x) = 0, \quad (4.2)$$

$$g_j(x) \geq 0, \quad j = 1, 2, \dots, J; \quad (4.3)$$

$$h_k(x) = 0, \quad k = 1, 2, \dots, K; \quad (4.4)$$

$$u_j g_j(x) = 0, \quad j = 1, 2, \dots, J; \quad (4.5)$$

$$u_j \geq 0, \quad j = 1, 2, \dots, J. \quad (4.6)$$

The multiplier  $u_j$  corresponds to the  $j$ -th inequality constraint and the multiplier  $v_k$  corresponds to the  $k$ -th equality constraint. Thus, there are a total number of  $J$  entries in the  $u$ -vector and  $K$  entries in the  $v$ -vector. The first equation arises from the optimality condition of the unconstrained Lagrangian function. The second and the third constraints are required for satisfying the constraints. The fourth equation arises only for inequality constraints. If the  $j$ -th inequality constraint is active at a point  $x$  (that is,  $g_j(x) = 0$ ), the product  $u_j g_j(x) = 0$ . On the other hand, if an inequality constraint is inactive at a point  $x$  (that is,  $g_j(x) > 0$ ), the Lagrange multiplier  $u_j$  is equal to zero, meaning thereby that in the neighbourhood of the point  $x$  the constraint has no effect on the optimum point. The final inequality condition suggests that in the case of active constraints, the corresponding Lagrange multiplier must be positive. This can be explained from a different perspective. By performing some algebraic manipulation, it can be shown that the Lagrange multiplier  $u_j$  is equal to the rate of change of the optimal function value with respect to the constant term of the  $j$ -th constraint (refer to Section 4.3). Thus, if the constant term of the  $j$ -th constraint is increased by one unit, the search space is more constrained and the optimal function value for the new constraint cannot be less than its previous value (in the case of minimization problems). Thus, the rate of change of the optimal function value for a positive change in constant term of any inequality constraint must always increase, which means that the Lagrange multiplier for every greater-than-or-equal type inequality constraint must be positive for minimization problems. By the same logic, we can conclude that  $u_j \leq 0$  for maximization problems.

A point  $x^{(t)}$  and two vectors  $u^{(t)}$  and  $v^{(t)}$  that satisfy all the above conditions are called Kuhn-Tucker points (also known as K-T points). There are a total of  $(N + 3J + K)$  Kuhn-Tucker conditions, as can be seen from Equations (4.2) to (4.6). In the above formulation, the variable bounds are also considered as inequality constraints. Thus, each variable bound  $x_i^{(L)} \leq x_i \leq x_i^{(U)}$  can be written in two inequality constraints as follows:

$$\begin{aligned} x_i - x_i^{(L)} &\geq 0, \\ x_i^{(U)} - x_i &\geq 0. \end{aligned}$$

In order to verify whether a point is a K-T point, all the above conditions are expressed in terms of  $u$  and  $v$  vectors. If there exists at least one set of  $u$  and  $v$  vectors, which satisfy all K-T conditions, the point is said to be a K-T point.

**EXERCISE 4.1.1**

Let us take an exercise problem to illustrate the Kuhn-Tucker points. We consider the following constrained Himmelblau function:

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

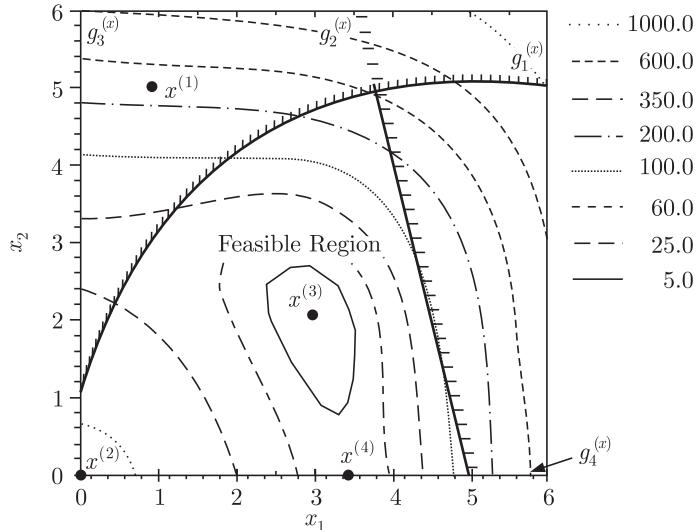
subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

$$x_1, x_2 \geq 0.$$

Notice that the objective function is the same as that used in exercise problems in Chapter 3. However, not every point in the search space is feasible. The feasible points are those that satisfy the above two constraints and variable bounds. Let us also choose four points  $x^{(1)} = (1, 5)^T$ ,  $x^{(2)} = (0, 0)^T$ ,  $x^{(3)} = (3, 2)^T$ , and  $x^{(4)} = (3.396, 0)^T$  to investigate whether each point is a K-T point. The feasible search space and these four points are shown on a contour plot of the objective function in Figure 4.1. The region on the other side of the hatched portion of a constraint line is feasible. The combination of two constraints and variable bounds makes the interior region feasible, as depicted in the figure. We would like to find out whether each of these points is a likely candidate for the minimum of the above NLP problem.



**Figure 4.1** The feasible search space and four points  $x^{(1)}$ ,  $x^{(2)}$ ,  $x^{(3)}$ , and  $x^{(4)}$ .

At first, we transform the variable bounds to two inequality constraints:  $g_3(x) = x_1 \geq 0$ , and  $g_4(x) = x_2 \geq 0$ . Thus, the above problem has four inequality constraints ( $J = 4$ ) and no equality constraint ( $K = 0$ ). There

are two problem variables:  $N = 2$ . Thus, for each point a total of  $(2 + 3 \times 4 + 0)$  or 14 Kuhn-Tucker conditions need to be checked. To formulate all K-T conditions, we first calculate the gradient of the objective function. In Table 4.1, we compute these gradients numerically and also compute the constraint values at all four points.

**Table 4.1** Gradient and Constraint Values at Four Different Points for Constrained Himmelblau's Function. [The gradients for constraints  $g_2$ ,  $g_3$ , and  $g_4$  are same for all points:  $\nabla g_2(x^{(t)}) = (-4, -1)^T$ ,  $\nabla g_3(x^{(t)}) = (1, 0)^T$ , and  $\nabla g_4(x^{(t)}) = (0, 1)^T$ .]

$t$	$x^{(t)}$	$g_1$	$g_2$	$g_3$	$g_4$	$\nabla f(x^{(t)})$	$\nabla g_1(x^{(t)})$
1	$\begin{pmatrix} 1 \\ 5 \end{pmatrix}$	-15.000	11.000	1.000	5	$\begin{pmatrix} 18 \\ 370 \end{pmatrix}$	$\begin{pmatrix} 8 \\ -10 \end{pmatrix}$
2	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	1.000	20.000	0.000	0	$\begin{pmatrix} -14 \\ -22 \end{pmatrix}$	$\begin{pmatrix} 10 \\ 0 \end{pmatrix}$
3	$\begin{pmatrix} 3 \\ 2 \end{pmatrix}$	18.000	6.000	3.000	2	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 4 \\ -4 \end{pmatrix}$
4	$\begin{pmatrix} 3.396 \\ 0 \end{pmatrix}$	23.427	6.416	3.396	0	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$	$\begin{pmatrix} 3.21 \\ 0 \end{pmatrix}$

Using the values in Table 4.1, we formulate the Kuhn-Tucker conditions in terms of a  $u$ -vector and investigate whether a feasible  $u$ -vector can be obtained by satisfying all conditions. If one such vector exists, then the chosen point is a K-T point. For the first point, the following conditions are obtained:

$$\begin{aligned} 18 - 8u_1 + 4u_2 - u_3 &= 0, & 370 + 10u_1 + u_2 - u_4 &= 0, \\ g_1(x^{(1)}) = -15 &\not\geq 0, & g_2(x^{(1)}) = 11 &> 0, \\ g_3(x^{(1)}) = 1 &> 0, & g_4(x^{(1)}) = 5 &> 0, \\ (-15)u_1 &= 0, & (11)u_2 &= 0, & (1)u_3 &= 0, & (5)u_4 &= 0, \\ u_1, u_2, u_3, u_4 &\geq 0. \end{aligned}$$

It is clear that the third condition is not satisfied. This is enough to conclude that the point  $x^{(1)}$  is not a K-T point. In fact, since the first constraint value is negative, this constraint is violated at this point and the point  $x^{(1)}$  is not a feasible point, as shown in Figure 4.1. If a point is infeasible, the point cannot be an optimal point.

For the second point  $x^{(2)} = (0, 0)^T$ , we obtain the following conditions:

$$\begin{aligned} -14 - 10u_1 + 4u_2 - u_3 &= 0, & -22 + u_2 - u_4 &= 0, \\ 1 > 0, & \quad 20 > 0, & 0 = 0, & \quad 0 = 0, \\ (1)u_1 = 0, & \quad (20)u_2 = 0, & (0)u_3 = 0, & \quad (0)u_4 = 0, \\ u_1, u_2, u_3, u_4 &\geq 0. \end{aligned}$$

All but the final set of conditions reveal that  $u_1 = 0$ ,  $u_2 = 0$ ,  $u_3 = -14$  and  $u_4 = -22$ . The final set of conditions are not satisfied with these values, because  $u_3$  and  $u_4$  are negative. Thus, the point  $x^{(2)}$  is not a K-T point. Since the constraints are not violated, the point is a feasible point, as shown in Figure 4.1. Thus, the point  $x^{(2)}$  cannot be an optimal point.

Similarly, the conditions for the third point  $x^{(3)} = (3, 2)^T$  are

$$\begin{aligned} -4u_1 + 4u_2 - u_3 &= 0, & 4u_1 + u_2 - u_4 &= 0, \\ 18 > 0, & \quad 6 > 0, & 3 > 0, & \quad 2 > 0, \\ (18)u_1 = 0, & \quad (6)u_2 = 0, & (3)u_3 = 0, & \quad (2)u_4 = 0, \\ u_1, u_2, u_3, u_4 &\geq 0. \end{aligned}$$

The vector  $u^* = (0, 0, 0, 0)^T$  satisfies all the above conditions. Thus, the point  $x^{(3)}$  is a K-T point (Figure 4.1). As mentioned earlier, K-T points are likely candidates for minimal points. To conclude, we may say that the optimality of a point requires satisfaction of more conditions, a point we shall discuss later.

The K-T conditions obtained for the point  $x^{(4)} = (3.396, 0)^T$  are

$$\begin{aligned} -3.21u_1 + 4u_2 - u_3 &= 0, & 1 + u_2 - u_4 &= 0, \\ 23.427 > 0, & \quad 6.416 > 0, & 3.396 > 0, & \quad 0 = 0, \\ (23.427)(u_1) = 0, & \quad (6.416)u_2 = 0, & (3.396)u_3 = 0, & \quad (0)u_4 = 0, \\ u_1, u_2, u_3, u_4 &\geq 0. \end{aligned}$$

The solution to the above conditions is the vector  $u^* = (0, 0, 0, 1)^T$ . Thus, the point  $x^{(4)}$  is also a K-T point. It is clear from the figure that the point  $x^{(3)}$  is the minimum point, but the point  $x^{(4)}$  is not a minimum point. Thus, we may conclude from the above exercise problem that a K-T point may or may not be a minimum point. But if a point is not a K-T point (point  $x^{(2)}$  or  $x^{(3)}$ ), then it cannot be an optimum point. In order to say more about optimality of points, the following two theorems are useful.

### Kuhn-Tucker necessity theorem

Consider the NLP problem described above. Let  $f$ ,  $g$ , and  $h$  be differentiable functions and  $x^*$  be a feasible solution to NLP. Let  $I = \{j|g_j(x^*) = 0\}$  denote

the set of active inequality constraints. Furthermore,  $\nabla g_j(x^*)$  for  $j \in I$  and  $\nabla h_k(x^*)$  for  $k = 1, 2, \dots, K$  are linearly independent (known as constraint qualification). If  $x^*$  is an optimal solution to NLP, there exists a  $(u^*, v^*)$  such that  $(x^*, u^*, v^*)$  satisfies Kuhn-Tucker conditions.

If a feasible point satisfies the constraint qualification condition, the K-T necessity theorem can be used to prove that the point is not optimal. However, if the constraint qualification is not satisfied at any point, the point may or may not be an optimal point. For a non-boundary yet feasible point, the constraint qualification condition depends on equality constraints only. In the absence of equality constraints, all feasible, non-boundary points meet the constraint qualification condition. The above theorem can only be used to conclude whether a point is not an optimum point. We use the above theorem to investigate the non-optimality of four points described before.

The first point is simply not a feasible point; thus the point cannot be an optimum point. At the second point, there are two active constraints:  $g_3(x^{(2)}) = 0$  and  $g_4(x^{(2)}) = 0$ . Since their derivatives  $(1, 0)^T$  and  $(0, 1)^T$  are linearly independent, they meet the constraint qualification. But since the point is not a K-T point (shown earlier), it cannot be an optimal point. The third point is a feasible as well as a non-boundary point. Since the point is a K-T point, we cannot conclude anything about it—the point could be or could not be a minimum point. The fourth point makes the constraint  $g_4(x)$  active. Thus, the constraint qualification is met. Again, the point is found to be a K-T point; thus we cannot conclude whether the point is a minimum or not.

The optimality of a Kuhn-Tucker point can be checked using the sufficiency theorem described below. However, the theorem is only applicable to a particular class of problems having a convex<sup>1</sup> objective function and concave<sup>2</sup> constraints.

The Hessian matrix of the Himmelblau function at  $x^{(0)} = (0, 0)^T$  is calculated in Step 4 of the first iteration in Exercise 3.4.3. The leading principal determinants are  $-42.114$  and  $1094.4$ . Since both of them are not positive, the Hessian matrix at  $x^{(0)}$  is not positive-definite nor positive-semidefinite. Thus, the Himmelblau function is not a convex function, as can also be seen from the contour plot. As a result, the sufficiency theorem cannot be applied to this function. We mention here that the Himmelblau function is chosen in this exercise problem simply because most algorithms presented in

<sup>1</sup>A function  $f(x)$  is defined as a convex function if for any two points  $x^{(1)}$  and  $x^{(2)}$  in the search space and for  $0 \leq \lambda \leq 1$

$$f(\lambda x^{(1)} + (1 - \lambda)x^{(2)}) \leq \lambda f(x^{(1)}) + (1 - \lambda)f(x^{(2)}).$$

The convexity of a function is tested by checking the Hessian matrix of the function. If the Hessian matrix is positive-definite or positive-semidefinite for all values of  $x$  in the search space, the function is a convex function (Strang, 1980).

<sup>2</sup>A function  $f(x)$  is defined as a concave function if the function  $-f(x)$  is a convex function.

this chapter are applied on this function. In order to maintain the continuity of our discussion, we present the sufficiency theorem and illustrate the usage of the theorem on a different problem with a convex objective function.

### Kuhn-Tucker sufficiency theorem

*Let the objective function be convex, the inequality constraints  $g_j(x)$  be all concave functions for  $j = 1, 2, \dots, J$  and equality constraints  $h_k(x)$  for  $k = 1, 2, \dots, K$  be linear. If there exists a solution  $(x^*, u^*, v^*)$  that satisfies the K-T conditions, then  $x^*$  is an optimal solution to the NLP problem.*

As shown earlier, Himmelblau's function is not a convex function. Thus, we cannot use the above theorem to conclude the optimality of points  $x^{(3)}$  and  $x^{(4)}$ . However, in such functions, if the number of K-T points to be investigated is small, the function values at those points can be compared to reduce the choice of candidates for the optimum point. In the above exercise problem, since  $f(x^{(3)}) = 0$  and  $f(x^{(4)}) = 33.535$ , the point  $x^{(3)}$  is a likely candidate of the true optimum point. It may be noted here that there exist similar theorems for testing optimum points in nonconvex objective functions. Interested readers may refer to a more advanced book (Mangasarian, 1969).

### EXERCISE 4.1.2

To illustrate the use of the above sufficiency theorem, we consider a convex objective function as follows:

$$f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 2)^2.$$

We construct an NLP problem with the above objective function and the same constraints and variable bounds as in Exercise 4.1.1. The objective function has the following Hessian matrix:

$$\nabla^2 f(x) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

The leading principal determinants are  $|2|$  and  $|\nabla^2 f(x)|$  or 2 and 4, respectively. Since both these values are positive, the Hessian matrix is positive-definite and the function  $f(x_1, x_2)$  is a convex function. Let us consider the point  $x^{(3)} = (3, 2)^T$ . The point  $x^{(3)}$  is a K-T point with a  $u$ -vector:  $u^* = (0, 0, 0, 0)^T$ . The first constraint  $g_1(x)$  is a concave function because the matrix

$$-\nabla^2 g_1(x) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

is a positive-definite matrix. The constraint  $g_2(x)$  is also a concave function because a linear function is both convex and concave. Thus, we can conclude that the point  $(3, 2)^T$  is a minimum point of the above constrained problem.

## 4.2 Lagrangian Duality Theory

We rewrite the optimization problem in the following form where  $f : \Re^n \rightarrow \Re$ , and  $\mathbf{g} : \Re^n \rightarrow \Re^m$ :

$$\begin{aligned} & \text{minimize} && f(\mathbf{x}), \\ & \text{subject to} && \mathbf{g}(\mathbf{x}) \leq 0, \\ & && \mathbf{x} \in X. \end{aligned} \tag{4.7}$$

Let the set  $X \subset \Re^n$  be a closed set; one can also have  $X = \Re^n$ . Equation (4.7) represents, in general, the *primal problem* (P). For simplicity, equality constraints are not considered here; however, with some modifications to our discussions here, they can be included as well. Let us denote the optimal solution to problem (P) be  $\mathbf{x}^P$  (primal solution) and the corresponding function value as  $f(\mathbf{x}^P)$ , or simply  $f^P$ . The corresponding vector denoting a Lagrange multiplier for each constraint is represented by  $\boldsymbol{\lambda}^P$  here.

The *Lagrangian dual problem* (D) is posed as follows (Bazaraa et al., 2004):

$$\begin{aligned} & \text{maximize} && \theta(\boldsymbol{\lambda}), \\ & \text{subject to} && \boldsymbol{\lambda} \in \Re_+^m, \\ & \text{where} && \theta(\boldsymbol{\lambda}) = \inf \{L(\mathbf{x}, \boldsymbol{\lambda}) : \mathbf{x} \in X\} \\ & && = \inf \left\{ f(\mathbf{x}) + \sum_{i=1}^m \lambda_i g_i(\mathbf{x}) : \mathbf{x} \in X \right\}, \end{aligned} \tag{4.8}$$

where  $L(\mathbf{x}, \boldsymbol{\lambda})$  is the Lagrangian function corresponding to (P), that is,  $L : \Re^n \times \Re_+^m \rightarrow \Re$ , with

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^m \lambda_i g_i(\mathbf{x}), \tag{4.9}$$

where  $\lambda_i$  is the dual variable corresponding to the  $i$ -th constraint. For the dual problem (D), components of  $\boldsymbol{\lambda}$ -vector are treated as variables. For an inequality constraint,  $\lambda_i$  is restricted to be nonnegative. The function  $\theta(\boldsymbol{\lambda})$ , defined above, is known as the Lagrangian dual function. Let us denote the solution to dual problem (D) as  $\boldsymbol{\lambda}^D$  (dual solution) and the objective of the dual problem as  $\theta^D$ . The  $\mathbf{x}$ -vector corresponding to dual solution  $(\boldsymbol{\lambda}^D)$  need not be feasible to the primal problem nor it be always identical to the primal solution  $\mathbf{x}^P$ . The *duality gap* ( $\rho$ ) is defined as the difference between the objective values of the primal and dual solutions, or  $\rho = (f^P - \theta^D)$ .

It is important to note that for any  $\mathbf{x}$  feasible to (P) and any  $\boldsymbol{\lambda}$  feasible to (D), that is  $\lambda_i \geq 0$ , the weak duality theorem states the following (Bazaraa et al., 2004; Bector et al., 2005; Rockafellar, 1996):

$$f(\mathbf{x}) \geq \theta(\boldsymbol{\lambda}). \tag{4.10}$$

The *strong duality* says

$$f(\mathbf{x}^P) \geq \theta(\boldsymbol{\lambda}^D), \quad (4.11)$$

thereby meaning that duality gap is always nonnegative or  $\rho \geq 0$ . A special case is as follows:

$$f(\mathbf{x}^P) = \theta(\boldsymbol{\lambda}^D). \quad (4.12)$$

In fact, weak duality is enough to provide a lower bound to (P) but it will not provide in general the infimal value unless Equation (4.12) holds. Thus, it is important to know when Equation (4.12) holds. The relation given in Equation (4.12) is usually called the strong duality relation. However, strong duality does not hold unless  $f$  is convex, each  $g_i$  is convex and  $X$  is a convex set.

Furthermore, if (P) is convex,  $\mathbf{x}^P$  is the solution to (P), and the Slater's constraint qualification holds, then there exists  $\bar{\boldsymbol{\lambda}} \in \Re_+^m$  such that

- (i)  $L(\mathbf{x}^P, \boldsymbol{\lambda}) \leq L(\mathbf{x}^P, \bar{\boldsymbol{\lambda}}) \leq L(\mathbf{x}, \bar{\boldsymbol{\lambda}})$ , for any  $\mathbf{x} \in X$  and  $\boldsymbol{\lambda} \in \Re_+^m$ .
- (ii)  $\lambda_i g_i(\mathbf{x}^P) = 0$ , for  $i = 1, 2, \dots, m$ .

The first condition is referred to as the saddle point condition and the second condition is called the complementary slackness condition. It is important to note that these conditions cannot be guaranteed if (P) is non-convex.

The above results motivate researchers to look for the solution of the Lagrangian dual problem (D) for several reasons:

- (i) The dual problem (D) is always concave, meaning that the negative of the Lagrangian dual function  $\theta(\boldsymbol{\lambda})$  is always a convex function. Thus, if the Lagrangian dual subproblem can be solved exactly, the dual problem is comparatively easier to solve, although the original primal problem, P, may be harder to optimize.
- (ii) If in a problem, the number of decision variables  $n$  is much greater than the number of constraints,  $m$ , that is,  $n \gg m$ , the dual formulation helps in changing the role of the decision variables and constraints. Thus, the dual problem has a smaller dimensional search space, although the computation of the objective function requires solution of another optimization problem.
- (iii) Due to the fact that duality gap is always nonnegative, the optimal dual function value provides a *lower bound* of the optimal primal function value. That is, if the dual problem can be solved, its optimal objective value will provide a lower bound to the optimal objective value of the primal problem.
- (iv) For problems having a zero duality gap, if the dual solution  $(\boldsymbol{\lambda}^D)$  exists, it is identical to the Lagrange multiplier vector corresponding to the primal solution. The Lagrange multipliers are useful in many ways, for example in performing a *sensitivity* analysis to the primal problem (Reklaitis et al., 1983; Rao, 1984) that have many useful practical significance.

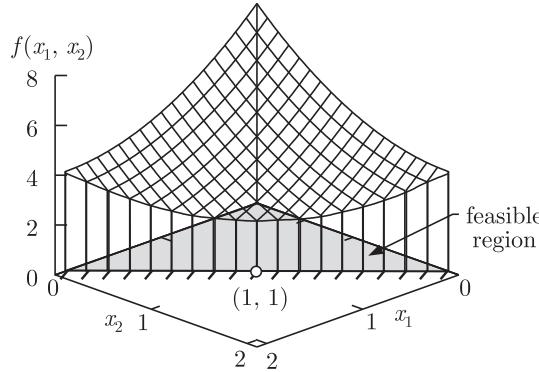
Let us illustrate the dual problem through an example primal problem:

$$\begin{aligned} \text{Minimize} \quad & f(x_1, x_2) = (x_1 - 2)^2 + (x_2 - 2)^2, \\ \text{Subject to} \quad & x_1 + x_2 \leq 2. \end{aligned}$$

Figure 4.2 shows the feasible region for the objective function values. Clearly, the optimum is at  $(x_1, x_2)^T = (1, 1)^T$  with a function value equal to 2. We find the Lagrange multiplier for the inequality constraint by writing the KKT conditions at the optimum point:

$$\begin{bmatrix} 2(x_1 - 2) \\ 2(x_2 - 2) \end{bmatrix}_{(1,1)^T} + \lambda \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

or,  $-2 + \lambda = 0,$   
or,  $\lambda = 2.$



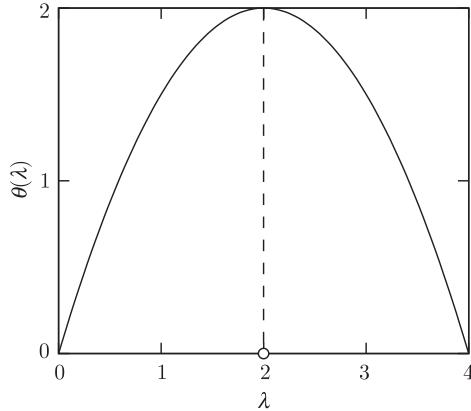
**Figure 4.2** Primal problem showing feasible region.

The corresponding dual problem is constructed below:

$$\begin{aligned} \text{Maximize } \theta(\lambda) = \min_{(x_1, x_2)} & ((x_1 - 2)^2 + (x_2 - 2)^2 + \lambda(x_1 + x_2 - 2)), \\ & \lambda \geq 0. \end{aligned} \tag{4.13}$$

Using the first and second-order optimality conditions on the minimization problem reveals the optimum  $x_1 = x_2 = 2 - \lambda/2$ . At this point,  $\theta(\lambda) = 2\lambda - \lambda^2/2$ . Maximizing this function reveals  $\lambda^* = 2$  and  $\theta(\lambda^*) = 2$ , which is identical to the optimal objective value at the primal solution. Figure 4.3 shows the dual function and its maximum point. Interestingly, every feasible  $\mathbf{x}$  has a higher function value than every feasible dual solution  $\lambda$ . Thus, both weak and strong duality conditions are satisfied at this problem.

With this background on the optimality conditions, we shall now present a number of optimization algorithms which attempt to find an optimum point iteratively. We begin with the direct search methods.



**Figure 4.3** Dual problem showing feasible region.

### 4.3 Transformation Methods

Transformation methods are the simplest and most popular optimization methods of handling constraints. The constrained problem is transformed into a sequence of unconstrained problems by adding penalty terms for each constraint violation. If a constraint is violated at any point, the objective function is penalized by an amount depending on the extent of constraint violation. Penalty terms vary in the way the penalty is assigned. Some penalty methods cannot deal with infeasible points at all and even penalize feasible points that are close to the constraint boundary. These methods are known as *interior penalty* methods. In these methods, every sequence of unconstrained optimization method finds a feasible solution using one of the methods described in Chapter 2. The solution found in one sequence is used as the starting solution for the next sequence of the unconstrained optimization. In subsequent sequences, the solution improves gradually and finally converges to the optimum solution.

The other kind of penalty methods penalize infeasible points but do not penalize feasible points. These methods are known as *exterior penalty* methods. In these methods, every sequence of unconstrained optimization finds an improved yet infeasible solution. There exist another method which penalizes both infeasible and feasible points. That method is known as the *mixed penalty* method. If the optimum solution is an interior point in the feasible region, one sequence of the exterior penalty method should find the minimum point. On the other hand, if the minimum is on a constraint boundary, several sequences may be necessary. In the following subsections, we describe two transformation methods.

#### 4.3.1 Penalty Function Method

Penalty function methods work in a series of sequences, each time modifying a set of penalty parameters and starting a sequence with the solution obtained

in the previous sequence. At any sequence, the following penalty function is minimized:

$$P(x, R) = f(x) + \Omega(R, g(x), h(x)), \quad (4.14)$$

where  $R$  is a set of penalty parameters,  $\Omega$  is the penalty term chosen to favor the selection of feasible points over infeasible points. For equality or inequality constraints, different penalty terms are used:

*Parabolic penalty*

$$\Omega = R\{h(x)\}^2.$$

This penalty term is used for handling equality constraints only. Since the feasible points always satisfy  $h(x) = 0$ , any infeasible point is penalized by an amount proportionate to the square of the constraint violation. The extent of penalty is controlled by fixing the parameter  $R$ . Since all infeasible points are penalized, this is an exterior penalty term. In the first sequence, a small value of  $R$  is used. In subsequent sequences,  $R$  is increased gradually.

*Infinite barrier penalty*

$$\Omega = R \sum_{j \in \bar{J}} |g_j(x)|.$$

This penalty term is used for handling inequality constraints. Here, the term  $R$  is a large number (usually a  $10^{20}$  is used) and  $\bar{J}$  denotes the set of violated constraints at the current point. Thus, a penalty proportionate to the constraint violation is added to the objective function. In this term, only one sequence with a large value of the penalty parameter is used. Since only infeasible points are penalized, this is also an exterior penalty term.

*Log penalty*

$$\Omega = -R \ln [g(x)].$$

This penalty term is also used for inequality constraints. For infeasible points,  $g(x) < 0$ . Thus, this penalty term cannot assign penalty to infeasible points. For feasible points, more penalty is assigned to points close to the constraint boundary or points with very small  $g(x)$ . Since only feasible points are penalized, this is an interior penalty term. In order to use this term, special care needs to be taken to handle infeasible points. Using this term, the first sequence is started with a large value of  $R$ . Thereafter, the penalty parameter  $R$  is gradually reduced to a small value.

*Inverse penalty*

$$\Omega = R \left[ \frac{1}{g(x)} \right].$$

Like the log penalty term, this term is also suitable for inequality constraints. This term penalizes only feasible points—the penalty is more for boundary points. This is also an interior penalty term and the penalty parameter is assigned a large value in the first sequence. In subsequent sequences, the parameter  $R$  is reduced gradually.

*Bracket operator penalty*

$$\Omega = R\langle g(x) \rangle^2,$$

where  $\langle \alpha \rangle = \alpha$ , when  $\alpha$  is negative; zero, otherwise. This term also handles inequality constraints. Since the bracket operator assigns a positive value to the infeasible points, this is an exterior penalty term. The first sequence begins with a small value of the penalty parameter  $R$  and is increased in subsequent sequences. This operator is mostly used in handling inequality constraints.

The change of penalty parameter  $R$  in successive sequences of the penalty function method depends on whether an exterior or an interior penalty term is used. If the optimum point of the unconstrained objective function is the true optimum of the constrained problem (that is, the constraints do not exclude the optimum point of the unconstrained objective function), an initial penalty parameter  $R = 0$  (or any other value of  $R$ ) will solve the constrained problem. Otherwise, if the constraints make the optimum of the unconstrained objective function infeasible, a number of sequences of the unconstrained optimization algorithm must be applied on a penalized objective function. When this happens, the constrained optimum point is usually a boundary point (a point that falls on at least one constraint surface). In the exterior penalty method, a feasible or an infeasible point can be used as the initial point of the first sequence, whereas in the interior penalty method, usually an initial feasible point is desired. In the case of exterior penalty term, a small initial value of  $R$  results in an optimum solution close to the unconstrained optimum point. As  $R$  is increased in successive sequences, the solution improves and finally approaches the true constrained optimum point. With the interior penalty term, however, a large initial value of  $R$  results in a feasible solution far away from the constraint boundaries. As  $R$  is decreased, the solution is improved and approaches the true optimum point.

In the following, we describe the penalty function method. A suitable penalty term can be used for any problem and depending on the penalty term used, the penalty parameter  $R$  needs to be modified.

### Algorithm

**Step 1** Choose two termination parameters  $\epsilon_1, \epsilon_2$ , an initial solution  $x^{(0)}$ , a penalty term  $\Omega$ , and an initial penalty parameter  $R^{(0)}$ . Choose a parameter  $c$  to update  $R$  such that  $0 < c < 1$  is used for interior penalty terms and  $c > 1$  is used for exterior penalty terms. Set  $t = 0$ .

**Step 2** Form  $P(x^{(t)}, R^{(t)}) = f(x^{(t)}) + \Omega(R^{(t)}, g(x^{(t)}), h(x^{(t)}))$ .

**Step 3** Starting with a solution  $x^{(t)}$ , find  $x^{(t+1)}$  such that  $P(x^{(t+1)}, R^{(t)})$  is minimum for a fixed value of  $R^{(t)}$ . Use  $\epsilon_1$  to terminate the unconstrained search.

**Step 4** Is  $|P(x^{(t+1)}, R^{(t)}) - P(x^{(t)}, R^{(t-1)})| \leq \epsilon_2$ ?

If yes, set  $x^T = x^{(t+1)}$  and **Terminate**;

Else go to Step 5.

**Step 5** Choose  $R^{(t+1)} = cR^{(t)}$ . Set  $t = t + 1$  and go to Step 2.

The main advantage of this method is that any constraint (convex or nonconvex) can be handled. The algorithm does not take into account the structure of the constraints, that is, linear or nonlinear constraints can be tackled with this algorithm. However, there is one difficulty with this method. At every sequence, the penalized function becomes somewhat distorted with respect to the original objective function. In Exercise 4.3.1, the distortion of the objective function in successive sequences of the penalty function method is illustrated. The distortion of the function causes the unconstrained search to become slow in finding the minimum of the penalized function, thereby taking more function evaluations to converge to the optimal solution. Moreover, in the presence of multiple constraints the transformation in Equation (4.14) can cause the penalized function to have artificial local optima, where the penalty function algorithm may get attracted to. Nevertheless, this algorithm is one of the popular methods to handle constraints.

### EXERCISE 4.3.1

Consider the constrained Himmelblau's function:

$$\text{Minimize } (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$(x_1 - 5)^2 + x_2^2 - 26 \geq 0, \quad x_1, x_2 \geq 0.$$

The inclusion of the constraint changes the unconstrained optimum point. The feasible region and the optimum point of this NLP is shown in Figure 4.4. The figure shows that the original optimum point  $(3, 2)^T$  is now an infeasible point. The new optimum is a point on the constraint line that touches a contour line at that point.

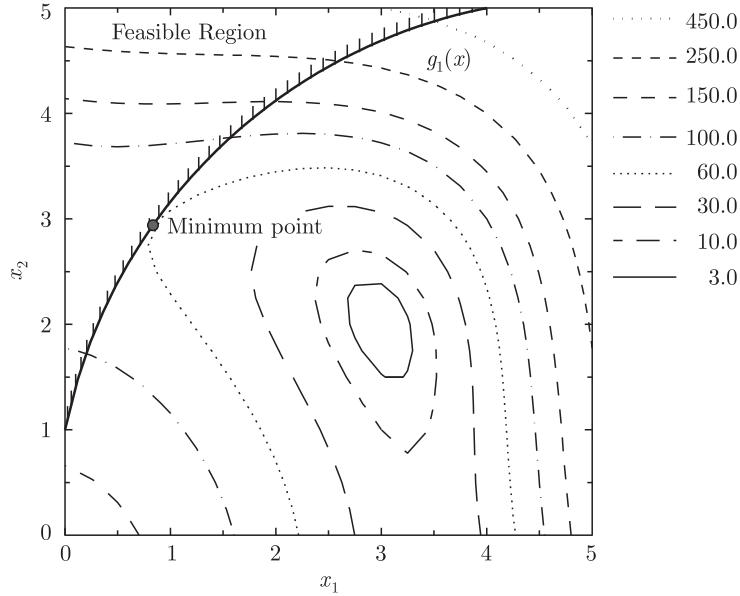
**Step 1** We use the bracket-operator penalty term to solve this problem. The bracket operator penalty term is an exterior penalty term. We choose an infeasible point  $x^{(0)} = (0, 0)^T$  as the initial point. We also choose a small value for the penalty parameter:  $R^{(0)} = 0.1$ . We choose two convergence parameters  $\epsilon_1 = \epsilon_2 = 10^{-5}$ .

**Step 2** The next task is to form the penalty function:

$$\begin{aligned} P(x, R^{(0)}) &= (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \\ &\quad + 0.1 \times ((x_1 - 5)^2 + x_2^2 - 26)^2. \end{aligned}$$

In the above formulation, the variable bounds must also be included as inequality constraints. For clarity and ease of illustration, we have not included the variable bounds in our formulation.

**Step 3** The above unconstrained function can now be minimized using one of the methods described in the previous chapter. Here, we use the steepest



**Figure 4.4** The feasible search space and the true minimum of the constrained problem in Exercise 4.3.1.

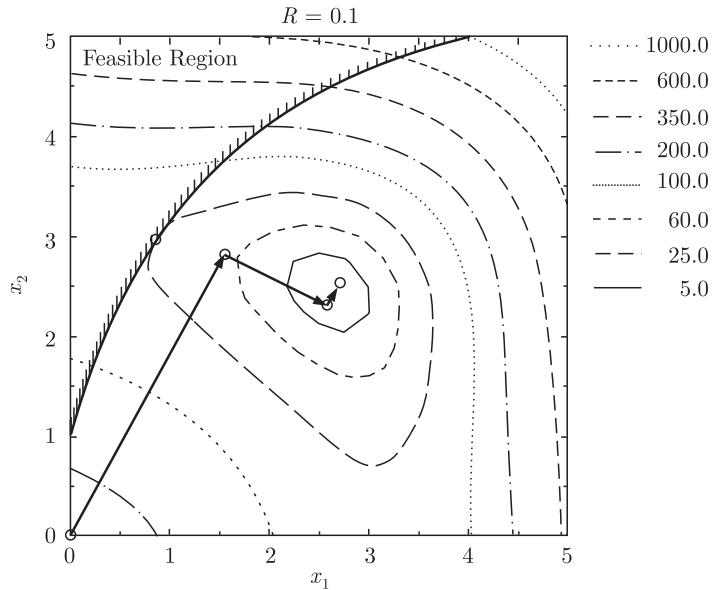
descent method to solve the above problem (the FORTRAN code presented at the end of Chapter 3 is used). We begin the algorithm with an initial solution  $x^{(0)} = (0, 0)^T$  having  $f(x^{(0)}) = 170.0$ . At this point, the constraint violation is  $-1.0$  and the penalized function value  $P(x^{(0)}, R^{(0)}) = 170.100$ . Intermediate points obtained by the steepest descent algorithm are tabulated in Table 4.3.1, and some of these points are shown in Figure 4.5. After 150 function evaluations, the solution  $x^* = (2.628, 2.475)^T$  having a function value equal to  $f(x^*) = 5.709$  is obtained. At this point, the constraint violation is equal to  $-14.248$ , but has a penalized function value equal to  $25.996$ , which is smaller than that at the initial point. Even though the constraint violation at this point is greater than that at the initial point, the steepest descent method has minimized the penalized function  $P(x, R^{(0)})$  from  $170.100$  to  $25.996$ . We set  $x^{(1)} = (2.628, 2.475)^T$  and proceed to the next step.

**Step 4** Since this is the first iteration, we have no previous penalized function value to compare with; thus we move to Step 5.

**Step 5** At this step, we update the penalty parameter  $R^{(1)} = 10 \times 0.1 = 1.0$  and move to Step 2. This is the end of the first sequence. It is important here to note that with a different initial point, we could have also converged to the same point. But simulation runs with certain initial points may have taken a longer time to converge than with other points. However, solutions in subsequent sequences will be identical for all simulations.

**Table 4.2** Tabulation of the Intermediate Points Obtained using the Steepest Descent Algorithm in Exercise 4.2.1

Sequence <i>t</i>	$R^{(t)}$	Solution $x^{(t)}$	$P(x^{(t)}, R^{(t)})$	Constraint violation
1	0.1	$(0, 0)^T$	170.100	-1.000
		$(2.569, 2.294)^T$	27.119	-14.828
		$(2.657, 2.455)^T$	26.019	-14.483
		$\vdots$	$\vdots$	$\vdots$
		$(2.628, 2.475)^T$	25.996	-14.248
2	1.0	$(2.628, 2.475)^T$	208.70	-14.248
		$(1.730, 3.412)^T$	75.140	-3.655
		$(1.166, 2.871)^T$	60.986	-3.058
		$\vdots$	$\vdots$	$\vdots$
3	10.0	$(1.011, 2.939)^T$	58.757	-1.450
		$(0.906, 3.016)^T$	60.530	-0.143
		$\vdots$	$\vdots$	$\vdots$
		$(0.844, 2.934)^T$	60.233	-0.119

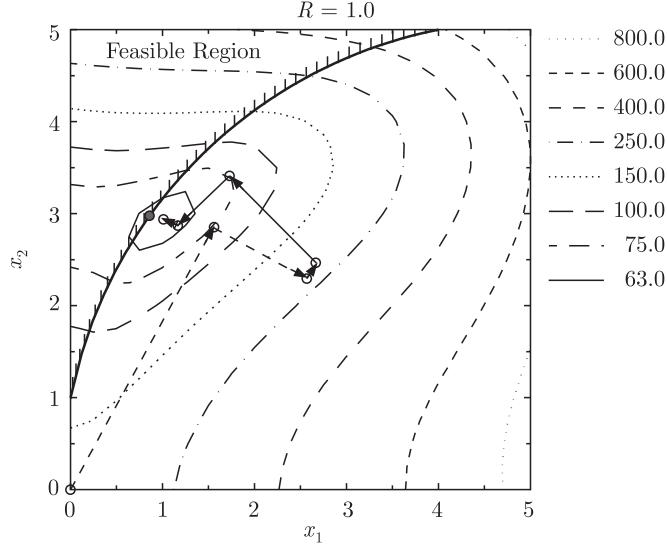


**Figure 4.5** A simulation of the steepest descent method on the penalized function with  $R = 0.1$ . The feasible region is marked. The hashes used to mark the feasible region are different from that in most other figures in this book.

**Step 2** The new penalized function in the second sequence is as follows:

$$\begin{aligned} P(x, R^{(1)}) &= (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \\ &\quad + 1.0 \times ((x_1 - 5)^2 + x_2^2 - 26)^2. \end{aligned}$$

**Step 3** At this step, we once again use the steepest descent method to solve the above problem from the starting point  $(2.628, 2.475)^T$ . Table 4.3.1 shows intermediate points of the simulation run. The minimum of the function is found after 340 function evaluations and is  $x^{(2)} = (1.011, 2.939)^T$ . At this point, the constraint violation is equal to  $-1.450$ , which suggests that the point is still an infeasible point. The penalized function and the minimum of the function are both shown in Figure 4.6. The progress of the previous



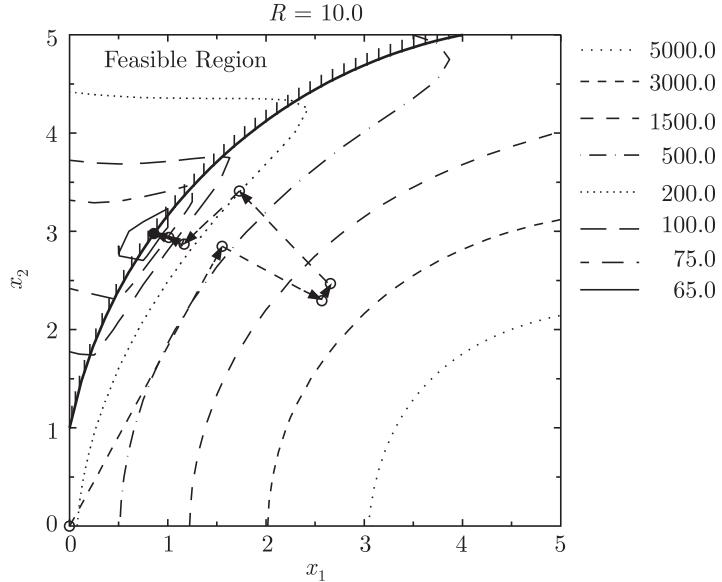
**Figure 4.6** Intermediate points using the steepest descent method for the penalized function with  $R = 1.0$  (solid lines). The hashes used to mark the feasible region is different from that in most other figures in this book.

sequence is also shown using dashed lines. Observe that this penalized function is distorted with respect to the original Himmelblau function. This distortion is necessary to shift the minimum point of the current function closer to the true constrained minimum point. Also notice that the penalized function at the feasible region is undistorted.

**Step 4** Comparing the penalized function values, we observe that  $P(x^{(2)}, 1.0) = 58.664$  and  $P(x^{(1)}, 0.1) = 25.996$ . Since they are very different from each other, we continue with Step 5.

**Step 5** The new value of the penalty parameter is  $R^{(2)} = 10.0$ . We increment the iteration counter  $t = 2$  and go to Step 2.

In the next sequence, the penalized function is formed with  $R^{(2)} = 10.0$ . The penalized function and the corresponding solution is shown in Figure 4.7. This time the steepest descent algorithm starts with an initial solution  $x^{(2)}$ . The minimum point of the sequence is found to be  $x^{(3)} = (0.844, 2.934)^T$  with a constraint violation equal to  $-0.119$ . Figure 4.7 shows the extent of distortion of the original objective function. Compare the contour levels



**Figure 4.7** Intermediate points obtained using the steepest descent method for the penalized function with  $R = 10.0$  (solid lines near the true optimum). Notice the distortion in the function. The hashes used to mark the feasible region are different from that in most other figures in this book.

shown at the top right corner of Figures 4.5 and 4.7. With  $R = 10.0$ , the effect of the objective function  $f(x)$  is almost insignificant compared to that of the constraint violation in the infeasible search region. Thus, the contour lines are almost parallel to the constraint line. Fortunately in this problem, the increase in the penalty parameter  $R$  only makes the penalty function steeper in the infeasible search region. In problems with a sufficiently nonlinear objective function and with multiple constraints, a large value of the penalty parameter may create one or more artificial local optima in the search space, thereby making it difficult for the unconstrained search to obtain the correct solution. The advantage of using the unconstrained search method sequentially is that the unconstrained search is always started from the best point found in the previous sequence. Thus, despite the presence of many local optima in the search space, the search at every sequence is initiated from a point near the correct optimum point. This makes it easier for the unconstrained search to find the correct solution.

After another sequence (iteration) of this algorithm, the obtained solution is

$$x^{(4)} = (0.836, 2.940)^T$$

with a constraint violation of only  $-0.012$ . This point is very close to the true constrained optimum solution. A few more iterations of the penalty function method may be performed to get a solution with the desired accuracy. Although a convergence check with a small difference in the penalized function

value at two consecutive sequences is used in this algorithm, any other convergence criteria (for example, a small difference in the  $x$ -vector of the solutions in two successive sequences) may also be used. At the end of this chapter, we present a FORTRAN code implementing the above penalty function method.

In the presence of multiple constraints, it is observed that the performance of the penalty function method improves considerably if the constraints and the objective functions are first normalized before constructing the penalized function. An inequality constraint  $g_j(x) \geq 0$  can be normalized as follows:

$$\frac{g_j(x)}{g_{\max}} \geq 0,$$

where  $g_{\max}$  is the maximum value of the constraint  $g_j(x)$  in the search space. Often, engineering design problems contain constraints restraining resource or capacity of  $b_j$  as  $g'_j(x) \leq b_j$ . The constraint can be normalized as follows:

$$1 - \frac{g'_j(x)}{b_j} \geq 0.$$

If an upper bound of the objective function is known, the objective function can also be normalized as shown above, but the normalization of the constraints is more important than that of the objective function.

### 4.3.2 Method of Multipliers

The problem with the penalty function method is that the penalized function becomes distorted at later sequences and that the unconstrained methods may face difficulty in optimizing those distorted functions. There exist a number of techniques to alleviate this problem. One method is to use a fixed penalty parameter  $R$  with a multiplier corresponding to each constraint. The constraint violation is increased by the multiplier value before calculating the penalty term. Thereafter, an equivalent term is subtracted from the penalty term. This method works in successive sequences, each time updating the multipliers in a prescribed manner. The penalty function is modified as follows:

$$\begin{aligned} P(x, \sigma^{(t)}, \tau^{(t)}) = & f(x) + R \sum_{j=1}^J \left\{ \left( \langle g_j(x) + \sigma_j^{(t)} \rangle \right)^2 - \left( \sigma_j^{(t)} \right)^2 \right\} \\ & + R \sum_{k=1}^K \left\{ \left( h_k(x) + \tau_k^{(t)} \right)^2 - \left( \tau_k^{(t)} \right)^2 \right\}, \end{aligned}$$

where  $\langle \cdot \rangle$  is the bracket operator as described in Section 4.3.1. The penalty parameter  $R$  is kept constant throughout a simulation run. The multipliers

$\sigma_j$  and  $\tau_k$  are updated in successive sequences as follows:

$$\sigma_j^{(t+1)} = \langle g_j(x^{(t)}) + \sigma_j^{(t)} \rangle, \quad j = 1, 2, \dots, J;$$

$$\tau_k^{(t+1)} = h_k(x^{(t)}) + \tau_k^{(t)}, \quad k = 1, 2, \dots, K.$$

By differentiating the term  $P(x, \sigma^{(t)}, \tau^{(t)})$ , it can be shown that the final solution  $x^T$  of the above procedure is a K-T point (Reklaitis et al., 1983). The above formulation does not distort the original objective function but shifts it towards the constrained optimum point. Thus, the complexity of solving the penalized function remains the same as that of the original objective function. Another advantage of this method is that the final values of the multipliers can be used to compute the corresponding Lagrange multipliers, which are also known as *shadow prices*, using the following two equations:

$$u_j = -2R\sigma_j^T, \quad (4.15)$$

$$v_k = -2R\tau_k^T. \quad (4.16)$$

The importance of Lagrange multipliers in the context of constrained optimization will be understood better, when we discuss sensitivity analysis in the next section.

The method of multiplier (MOM) is similar to the penalty function method. The initial values of multipliers ( $\sigma_j^{(0)}$  and  $\tau_k^{(0)}$ ) are usually kept to be zero. The penalty parameter  $R$  is kept constant in all sequences. At every sequence, the unconstrained function is minimized and a new point is found. The multipliers are updated using the constraint value at the new point and a new unconstrained function is formed. This process continues until a convergence criterion is met.

### Algorithm

**Step 1** Choose a penalty parameter  $R$ , termination parameters  $\epsilon_1$  and  $\epsilon_2$ . Choose an initial solution  $x^{(0)}$ . Set multipliers  $\sigma_j^{(0)} = \tau_k^{(0)} = 0$  and the iteration counter  $t = 0$ .

**Step 2** Next, form the penalized function:

$$\begin{aligned} P(x, \sigma^{(t)}, \tau^{(t)}) &= f(x) + R \sum_{j=1}^J \left\{ \left( \langle g_j(x) + \sigma_j^{(t)} \rangle \right)^2 - \left( \sigma_j^{(t)} \right)^2 \right\} \\ &\quad + R \sum_{k=1}^K \left\{ \left( h_k(x) + \tau_k^{(t)} \right)^2 - \left( \tau_k^{(t)} \right)^2 \right\}. \end{aligned}$$

**Step 3** Use an unconstrained optimization technique to solve the penalized function  $P(x, \sigma^{(t)}, \tau^{(t)})$  from the starting point  $x^{(t)}$  with a convergence factor  $\epsilon_1$ . During this optimization,  $\sigma^{(t)}$  and  $\tau^{(t)}$  are kept fixed; only the vector  $x$  is varied. Let us say the solution is  $x^{(t+1)}$ .

**Step 4** Is  $|P(x^{(t+1)}, \sigma^{(t)}, \tau^{(t)}) - P(x^{(t)}, \sigma^{(t-1)}, \tau^{(t-1)})| \leq \epsilon_2$ ?

If yes, set  $x^T = x^{(t+1)}$  and **Terminate**;

Else go to Step 5.

**Step 5** Update  $\sigma_j^{(t+1)} = \langle g_j(x^{(t)}) + \sigma_j^{(t)} \rangle$  for all  $j = 1, 2, \dots, J$  and  $\tau_k^{(t+1)} = h_k(x^{(t)}) + \tau_k^{(t)}$  for all  $k = 1, 2, \dots, K$ . Set  $t = t + 1$  and go to Step 2.

In the following, we present hand simulation of a few iterations of the method of multiplier technique to a numerical optimization problem.

### EXERCISE 4.3.2

Consider the following function:

$$\text{Minimize} \quad (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$(x_1 - 5)^2 + x_2^2 - 26 \geq 0,$$

$$x_1, x_2 \geq 0.$$

**Step 1** We choose  $R = 0.1$  and an initial point  $x^{(0)} = (0, 0)^T$ . We set  $\sigma_1^{(0)} = 0$ . We also choose convergence factors  $\epsilon_1 = \epsilon_2 = 10^{-5}$ . We set an iteration counter  $t = 0$ . Once again, we simplify our calculations by not considering the variable bounds as inequality constraints.

**Step 2** The unconstrained function becomes as follows:

$$\begin{aligned} P(x, \sigma_1^{(0)}) &= (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \\ &\quad + 0.1 \times \langle (x_1 - 5)^2 + x_2^2 - 26 \rangle^2. \end{aligned}$$

Recall that the bracket operator  $\langle \alpha \rangle$  takes a nonzero value  $\alpha$ , if  $\alpha$  is negative. Since this function and the initial point considered here are the same as in the first iteration of Exercise 4.3.1, Steps 3 and 4 are not repeated here. We simply reproduce the solution  $x^{(1)} = (2.628, 2.475)^T$  (Table 4.3.2) and move to Step 5.

**Step 5** At this step, the constraint violation is equal to  $g_1(x^{(1)}) = -14.248$ . Thus, we update the multiplier

$$\sigma_1^{(1)} = \langle -14.248 \rangle + 0 = -14.248.$$

We set  $t = 1$  and proceed to Step 2. This completes one sequence of the MOM algorithm.

**Step 2** At this step, the unconstrained function is as follows:

$$\begin{aligned} P(x, \sigma_1^{(1)}) &= (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 + 0.1 \\ &\times \left\{ ((x_1 - 5)^2 + x_2^2 - 26 - 14.248) \right\}^2 - (-14.248)^2 \end{aligned}$$

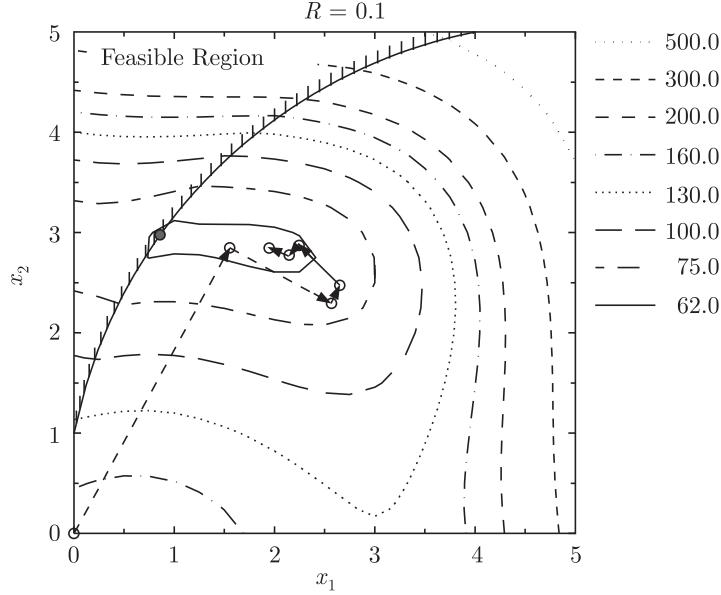
**Step 3** Starting from the point  $x^{(1)} = (2.628, 2.475)^T$ , we use the unconstrained steepest descent search method to solve the above function. We obtain the solution  $x^{(2)} = (1.948, 2.846)^T$ . The intermediate points of the steepest descent method are shown in Table 4.3.2.

**Table 4.3** Tabulation of the Intermediate Points Obtained using the Steepest Descent Algorithm for the Method of Multiplier (MOM) Algorithm. (The first sequence is the same as that in the penalty function method.)

Sequence $t$		Solution $x^{(t)}$	$P(x^{(t)}, R^{(t)})$	Constraint violation
1	0.1	$(0, 0)^T$	170.100	-1.000
		$(2.569, 2.294)^T$	27.119	-14.828
		$(2.657, 2.455)^T$	26.019	-14.483
		$\vdots$	$\vdots$	$\vdots$
		$(2.628, 2.475)^T$	25.996	-14.248
2	0.1	$(2.628, 2.475)^T$	66.597	-14.248
		$(2.246, 2.872)^T$	61.030	-10.167
		$(2.147, 2.776)^T$	60.478	-10.154
		$\vdots$	$\vdots$	$\vdots$
		$(1.948, 2.846)^T$	60.129	-8.586
3	0.1	$(1.948, 2.846)^T$	74.872	-8.586
		$(1.606, 3.165)^T$	71.373	-4.464
		$(1.376, 2.918)^T$	68.425	-4.352
		$\vdots$	$\vdots$	$\vdots$
		$(0.932, 3.074)^T$	61.245	-0.002

At this point, the objective function value is  $f(x^{(2)}) = 28.292$  and the constraint violation is  $g_1(x^{(2)}) = -8.586$ , an improvement from the first iteration. Figure 4.8 shows the contour of the penalized function and the progress of the steepest descent method. It is clear that the function is less distorted as compared to that in the penalty function method. In fact, this

function can be viewed as a shift of the original penalized function (Figure 4.5) towards the current minimum point. Note that the function does not change in the feasible region.



**Figure 4.8** Intermediate points obtained using the steepest descent method for the minimization of  $P(x, \sigma_1^{(1)})$ . The hashes used to mark the feasible region are different from that in most other figures in this book.

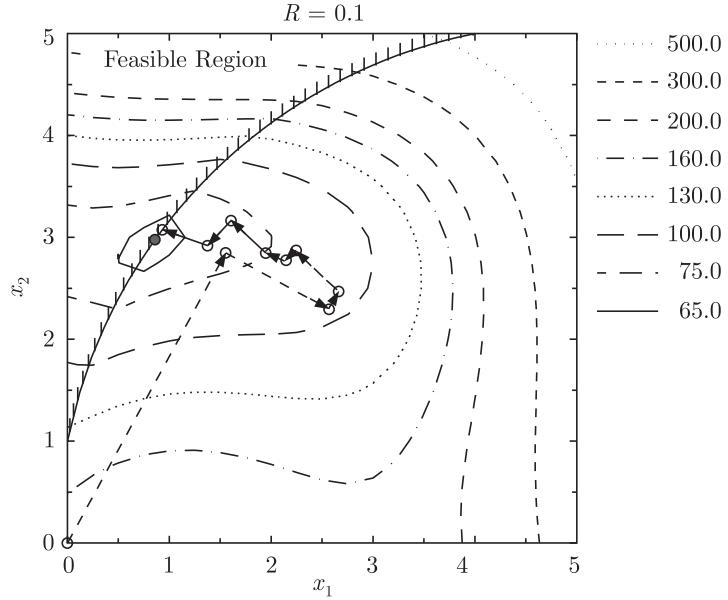
**Step 4** Comparing the penalized function values, we observe that  $P(x^{(2)}, \sigma_1^{(1)}) = 60.129$  and  $P(x^{(1)}, \sigma_1^{(0)}) = 58.664$ . Since the difference in these quantities is large compared to the chosen  $\epsilon_2$ , we move to Step 5.

**Step 5** We update the multiplier as follows:

$$\sigma_1^{(2)} = \langle g_1(x^{(2)}) + \sigma_1^{(1)} \rangle = -8.586 - 14.248 = -22.834.$$

We increment the counter  $t = 2$  and move to Step 2.

Another iteration of this method yields a solution  $x^{(3)} = (0.932, 3.074)^T$  with a function value equal to 61.241. At this point, the constraint violation is only  $g_1(x^{(3)}) = -0.002$ . Figure 4.9 shows the contour plot of the unconstrained function at the end of this iteration. The new value of the multiplier is  $\sigma_1^{(3)} = -22.836$ . We also observe that the penalized function  $P(x, \sigma_1^{(2)})$  causes a result of translation of the original objective function in the infeasible region, whereas the feasible search space does not get modified at all. This can be observed from the figure as well, where no substantial distortion is found at the top-left corner of the plot. The contour levels shown at the top right corner is similar to that in Figure 4.5. This property of the



**Figure 4.9** Intermediate points obtained at the third iteration of the MOM algorithm. The penalized function is not distorted, but translated towards the constrained optimum point. The hashes used to mark the feasible region are different from that in most other figures in this book.

penalized functions in the MOM improves the convergence characteristics of the unconstrained optimization methods.

A few more iterations of this algorithm find the true constrained optimum point  $(0.829, 2.933)^T$  with the desired accuracy.

#### 4.4 Sensitivity Analysis

When the optimum solution of a problem is obtained using an optimization method, a common question arises: Is the obtained solution really optimal? As discussed earlier, for a certain class of NLP problems, the necessity and sufficient theorems can be used to answer the above question. But for other problems, there is no definitive answer to this question. However, there is a practical answer to this question. In most real-world optimization problems the optimum solution is not known, but usually a reasonable solution (either obtained from the previous studies or from intuition) is available. In most cases, the effort of using the optimization process is justified, if a solution better than the previously known solution is found. The optimality of the obtained solution is further bolstered by applying the optimization algorithm a number of times (usually five to ten times) from different initial solutions. If at each time the obtained solution is the same, it is concluded that the solution is an optimum point.

Once the designer is more or less convinced that an optimal solution is found, the following question arises: How does the optimal value of the objective function change if some problem parameters<sup>3</sup> are changed marginally? This question is more relevant in problems having resource constraints. For example, consider an optimization problem, formulated based on a maximum deflection constraint of 5 mm at a specified point in a truss structure. Once the optimum solution is found, the designer may want to know the new optimal objective function value if the maximum deflection constraint is relaxed to 6 mm. One way to know the solution is to simulate the revised optimal problem again, but this method may be computationally expensive. However, a quick estimate of the solution can be achieved by using the optimal Lagrange multipliers corresponding to the constraint. This study of the effect of problem parameters on the optimal objective function value is known as *sensitivity analysis*. This analysis is usually performed after the optimal solution of the original problem has been found. In the case of multiple constraints, sensitivity analysis can be used to identify the critical constraint at which the optimal objective function depends the most. (Among all constraints, the Lagrange multiplier corresponding to the critical constraint has the maximum absolute value.)

The problem parameters mentioned in the above discussion are the important parameters in the design but are not the design variables. They are usually fixed at the time of formulation of the optimization problem and are kept constant during the simulation phase. On the other hand, the design variables are not fixed at the formulation phase and are varied during the simulation phase. Thus, the optimization process and the sensitivity analysis are complimentary to each other in the sense that the study of the effect of the design variables is the optimization process and the study of the effect of the problem parameters is the sensitivity analysis. To cite an example, consider a two-variable geometric optimization problem. Let us imagine that a two-dimensional region is bounded by a circle ( $x_1^2 + x_2^2 \leq a$ ) and straight line ( $x_1 + x_2 \geq b$ ). The objective of the optimization problem is to find the minimum distance of the bounded region from a point  $(c, d)$ . The problem can be formulated as an optimization problem as follows:

$$\left. \begin{array}{l} \text{Minimize} \quad (x_1 - c)^2 + (x_2 - d)^2 \\ \text{subject to} \\ \quad -x_1^2 - x_2^2 + a \geq 0, \\ \quad x_1 + x_2 - b \geq 0. \end{array} \right\} \quad (4.17)$$

In the above problem, the problem parameters are  $a$ ,  $b$ ,  $c$ , and  $d$ . The design variables are  $x_1$  and  $x_2$ . In order to solve the above problem using

---

<sup>3</sup>The problem parameters are important design parameters except the chosen design variables ( $x_i$ ).

an optimization algorithm, some definite values of all problem parameters are necessary. Fixing the problem parameters fixes the enclosed region exactly. For a given point  $(c, d)$ , the optimum minimum point  $(x_1^*, x_2^*)$  and the minimum distance can be found. Let us now make the problem more interesting and provide some flexibility on the part of the user. The user may want to know how this minimum distance changes if the bounding region is modified by changing either  $a$ ,  $b$ , or both. One way to achieve this change is to perform another simulation of the optimization algorithm on the revised problem. If the changes in  $a$  and  $b$  are large, this is the only way to find the new distance. But if the changes in  $a$  and  $b$  values are small, performing another optimization simulation is computationally expensive. It turns out that in this case there exists a relationship between the change of problem parameters (constants associated with constraints) and the corresponding optimal objective function values in terms of the Lagrange multipliers associated with the constraints. The method allows an easier way to find the new optimal objective function value without performing any extra function evaluations. But before we discuss that relationship, we present a method to compute the Lagrange multipliers.

In Equation (4.2), Kuhn-Tucker conditions are described. Recall that the variables  $u_j$  and  $v_k$  are the Lagrange multipliers corresponding to  $j$ -th inequality constraint and  $k$ -th equality constraint, respectively. Recall also that for greater-than-equal type inequality constraints, optimal values of the Lagrange multipliers  $u_j$  are nonnegative. However, Lagrange multipliers  $v_k$  can take any sign. At any given point, these multipliers can be computed by satisfying all conditions given in Equation (4.2). Since this involves  $(N + 3J + K)$  conditions, these computations are usually expensive. The Lagrange multipliers can also be computed using the MOM technique described in the previous section. The MOM technique not only finds the optimum solution but also computes the multipliers  $(\sigma$  and  $\tau$ ) which can be converted into corresponding Lagrange multipliers using Equations (4.15) and (4.16). Due to the numerical inaccuracies associated with the iterative optimization algorithms, the Lagrange multipliers may not be obtained with high precision. However, it may be noted here that the computation of Lagrange multipliers using the MOM technique is not always possible, especially in problems having an objective function with multiple optima. In those cases, the Lagrange multipliers can be found by considering them as design variables (along with design variable vector  $x$ ) in an optimization problem formulated using Kuhn-Tucker conditions:

$$\text{Minimize } \|\nabla f(x) - \sum_{j=1}^J u_j \nabla g_j(x) - \sum_{k=1}^K v_k \nabla h_k(x)\|,$$

subject to

$$g_j(x) \geq 0, \quad j = 1, 2, \dots, J;$$

$$h_k(x) = 0, \quad k = 1, 2, \dots, K;$$

$$u_j g_j(x) = 0, \quad j = 1, 2, \dots, J;$$

$$u_j \geq 0, \quad j = 1, 2, \dots, J.$$

Once the optimal Lagrange multipliers are found, the sensitivity analysis can be performed using the following procedure. The NLP problem given in Equation (4.1) can be rewritten by separating the constant term from the rest of the expression as follows:

$$\left. \begin{array}{l} \text{Minimize } f(x) \\ \text{subject to } g'_j(x) \geq \alpha_j, \quad j = 1, 2, \dots, J; \\ h'_k(x) = \beta_k, \quad k = 1, 2, \dots, K, \end{array} \right\} \quad (4.18)$$

where  $g_j(x) = g'_j(x) - \alpha_j$  and  $h_k(x) = h'_k(x) - \beta_k$ . By differentiating the objective function  $f(x)$  with respect to the problem parameters ( $\alpha_j$  and  $\beta_k$ ) at the optimum point, the following relationships are obtained (Reklaitis et al., 1983):

$$\frac{\partial f^*}{\partial \alpha_j} = u_j, \quad j = 1, 2, \dots, J; \quad (4.19)$$

$$\frac{\partial f^*}{\partial \beta_k} = v_k, \quad k = 1, 2, \dots, K. \quad (4.20)$$

The net change in the optimal objective function value is then obtained as follows:

$$\begin{aligned} \Delta f^* &= \sum_{j=1}^J \frac{\partial f^*}{\partial \alpha_j} \Delta \alpha_j + \sum_{k=1}^K \frac{\partial f^*}{\partial \beta_k} \Delta \beta_k, \\ &= \sum_{j=1}^J u_j \Delta \alpha_j + \sum_{k=1}^K v_k \Delta \beta_k. \end{aligned} \quad (4.21)$$

The above analysis is valid for small changes in right-side parameters and can be used to get an estimate in the change in value of optimal objective function value due to changes in right-side parameters without performing another optimization run.

We illustrate the sensitivity analysis procedure with the help of the following exercise problem.

#### EXERCISE 4.4.1

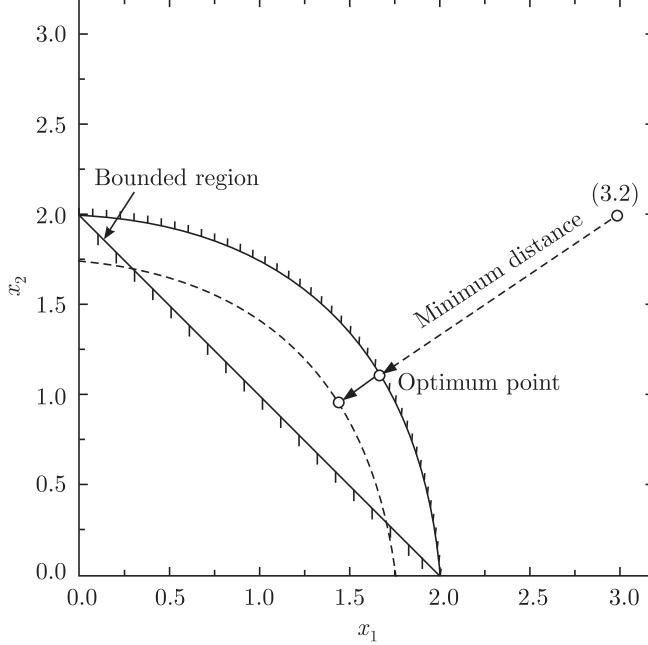
We consider the geometric optimization problem:

$$\text{Minimize } (x_1 - 3)^2 + (x_2 - 2)^2$$

$$\text{subject to } -x_1^2 - x_2^2 + 4 \geq 0,$$

$$x_1 + x_2 - 2 \geq 0.$$

Using Equation (4.2), we obtain the exact solution to the above problem:  $x^* = (1.664, 1.109)^T$  and  $f(x^*) = 2.579$ . The corresponding Lagrange multipliers are  $u_1^* = 0.803$  and  $u_2^* = 0$ . Figure 4.10 shows the bounded region and the minimum distance of the region from the point  $(3, 2)^T$ . Though we have found the exact optimal solution from geometric



**Figure 4.10** The bounded region and the point  $(3, 2)^T$ . The optimum point and the minimum distance of the bounded region from the point are also shown. The plot also shows how the minimum distance changes with the change in one of the constraints.

considerations, we use the MOM technique to find this optimum solution. In addition to finding the optimum solution, the MOM technique will also find the corresponding Lagrange multipliers. In the simulation of MOM, a penalty parameter  $R = 0.1$  is used. The obtained solution and the multipliers at each iteration of the MOM algorithm are shown in Table 4.4.

We begin with  $\sigma_1^{(0)} = \sigma_2^{(0)} = 0$ . After 10 iterations, we observe that the obtained solution is close to the true optimum solution. At this iteration, the Lagrange multiplier values are  $\sigma_1^{(10)} = -4.010$  and  $\sigma_2^{(10)} = 0$ . Assuming  $\sigma_1^T = -4.010$  and  $\sigma_2^T = 0$ , we calculate the corresponding Lagrange multipliers using Equations (4.15) and (4.16):

$$u_1 = -2R\sigma_1^T = -2(0.1)(-4.010) = 0.802,$$

$$u_2 = -2R\sigma_2^T = -2(0.1)(0) = 0,$$

**Table 4.4** Tabulation of the Intermediate Points  
Obtained by using the MOM Algorithm  
in Exercise 4.4.1

Iteration <i>t</i>	Solution $x^{(t)}$	Multipliers	
		$\sigma_1^{(t)}$	$\sigma_2^{(t)}$
0	(5.000, 5.000) <sup>T</sup>	0	0
1	(2.076, 1.384) <sup>T</sup>	-2.226	0
2	(1.844, 1.229) <sup>T</sup>	-3.135	0
3	(1.751, 1.167) <sup>T</sup>	-3.565	0
4	(1.708, 1.139) <sup>T</sup>	-3.781	0
5	(1.687, 1.125) <sup>T</sup>	-3.893	0
6	(1.676, 1.117) <sup>T</sup>	-3.950	0
7	(1.670, 1.113) <sup>T</sup>	-3.977	0
8	(1.668, 1.112) <sup>T</sup>	-3.996	0
9	(1.668, 1.110) <sup>T</sup>	-4.004	0
10	(1.665, 1.110) <sup>T</sup>	-4.010	0

which are also close to the true Lagrange multipliers found using the Kuhn-Tucker conditions. With these values of Lagrange multipliers, let us now investigate the effect of each constraint on the optimal objective function value by using the sensitivity analysis procedure and by referring to the bounded region shown in Figure 4.10. Since  $u_1$  is greater than  $u_2$ , the effect of the first constraint on the optimal objective function value is more. This aspect can also be visualized in Figure 4.10. Comparing this NLP problem with the problem stated in Equation (4.18), we observe that  $\alpha_1 = -4$  and  $\alpha_2 = 2$ . A small change in the value of  $\alpha_1$  changes the outcome of the optimum solution. On the other hand, a small change in the value of  $\alpha_2$  does not change the outcome of the optimum solution, because  $u_2 = 0$ . Thus, the first constraint is more crucial than the second one. Let us now compute the net change in the optimal objective function value for small changes in  $\alpha_1$  and  $\alpha_2$  values. If  $\alpha_1$  is now changed to  $-3$  and  $\alpha_2$  is unchanged, the changes are  $\Delta\alpha_1 = -3 - (-4) = 1$  and  $\Delta\alpha_2 = 0$ . Thus, the net change in optimal function value according to Equation (4.21) is

$$\Delta f^* = (0.802)(1) + (0)(0) = 0.802.$$

Therefore, the optimal function value increases, which means that we now have a larger distance from the point (3, 2). A change of the constraint from  $4 - x_1^2 - x_2^2 \geq 0$  to  $3 - x_1^2 - x_2^2 \geq 0$  (shown with a dashed line in the figure) takes the bounded region away from the point (3, 2) (as shown by the dashed curve line in the figure), thereby increasing the minimum distance of the

new bounded region from the fixed point  $(3, 2)^T$ . In fact, the new optimum function value is 3.510 which is 0.931 units more than the original optimal function value. This value is closer to the estimate  $\Delta f^*$  obtained using the sensitivity analysis.

## 4.5 Direct Search for Constrained Minimization

In the optimization methods discussed so far, the functional form of the constraints is not exploited to drive the search process, instead the search is governed only by the amount of constraint violation. In the direct search methods for constrained minimization, detailed structure of the constraints is exploited. Although this makes the search methods more efficient but restrictive to only specific classes of problems. One advantage of using a direct search method is that discontinuous and non-differentiable functions can be handled. Later, we shall discuss a number of gradient-based methods which use gradient information of the objective function as well as constraints to drive the search process. The direct search methods described in this section are all heuristic in nature. Thus, most of these methods do not have any convergence proof.

The generic structure of these algorithms is that they start with a feasible point. A new point is created using a fixed transition rule from the chosen initial point. If the new point is infeasible or inferior to the previous point, the point is not accepted and another point is found either at random or by using another transition rule. If the new point is feasible and better than the previous point, the point is accepted and the next iteration is performed from the new point. This process continues until a termination criterion is satisfied. In the following subsections, we describe three different direct search algorithms.

### 4.5.1 Variable Elimination Method

The variable elimination method is suitable for handling equality constraints only. In an optimization problem with  $K$  equality constraints ( $K < N$ ),  $K$  variables can be theoretically eliminated by using all  $K$  equality constraints. The variables to be eliminated are known as dependent variables ( $\hat{x}$ ) and the remaining variables are known as independent variables ( $\bar{x}$ ). The variables may be eliminated either explicitly or implicitly. Explicit elimination of variables is not always possible. Thus, implicit elimination of variables is usually adopted.

Let us assume that we would like to solve the following NLP problem having equality constraints only:

$$\begin{aligned} & \text{Minimize } f(x) \\ \text{subject to } & h_k(x) = 0, \quad k = 1, 2, \dots, K; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \dots, N. \end{aligned}$$

At first, the dependent variables  $\hat{x}$  are chosen from the set of  $N$  variables ( $x \equiv (\bar{x}, \hat{x})$ ). Then, they can be written in terms of the independent

variables. This takes care of the constraints. Thereafter, an initial feasible solution of independent variables  $\bar{x}^{(0)}$  is chosen. Since the constraints are implicitly satisfied, an unconstrained minimization technique described in Chapters 2 and 3 can be used to find the minimum point corresponding to independent variables. Once the optimal independent variables are found, the corresponding optimal dependent variables can be found using the equality constraints again. Even though the algorithm is simple, there is an important point to ponder. Since the dependent variables are eliminated from consideration in the unconstrained optimization, the original variable bounds for the independent variables may require a change. The objective function is a function of all  $N$  variables; thus the limits on dependent variables restrict the bounds of the independent variables. Thus, the unconstrained minimization problem becomes

$$\text{Minimize } f(\bar{x}, \hat{x}(\bar{x}))$$

in the search space  $\bar{x}_i^{(L')} \leq \bar{x}_i \leq \bar{x}_i^{(U')}$  for  $i = 1, 2, \dots, (N - K)$ . It is important here to note that although the new interval for the independent variables cannot be larger than the original interval. In other words, for any independent variable,  $x_i^{(L')} \geq x_i^{(L)}$  and  $x_i^{(U')} \leq x_i^{(U)}$ . The dependent variables  $\hat{x}(\bar{x}^{(t)})$  can be computed for a given values of independent variables  $\bar{x}^{(t)}$  by solving all equality constraints or by solving the following minimization problem for variables  $\hat{x}$ :

$$\text{Minimize } \sum_{k=1}^K \text{abs}[h_k(\bar{x}^{(t)}, \hat{x})], \quad \hat{x}_i^{(L)} \leq \hat{x}_i \leq \hat{x}_i^{(U)}.$$

The nomenclature used in the above description may look intimidating, but this algorithm is one of the simplest algorithms we have discussed in this chapter. We take an exercise problem to illustrate this method.

#### EXERCISE 4.5.1

Consider the constrained Himmelblau function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$h(x) = 26 - (x_1 - 5)^2 - x_2^2 = 0,$$

$$0 \leq x_1, x_2 \leq 5.$$

We describe the procedures of using both implicit and explicit variable elimination methods on this problem. Since there are two variables and one equality constraint, the dependent and independent variables are one each. Let us assume that we choose the second variable to be the dependent variable. Thus, according to our notation,  $\bar{x} = \{x_1\}$  and  $\hat{x} = \{x_2\}$ .

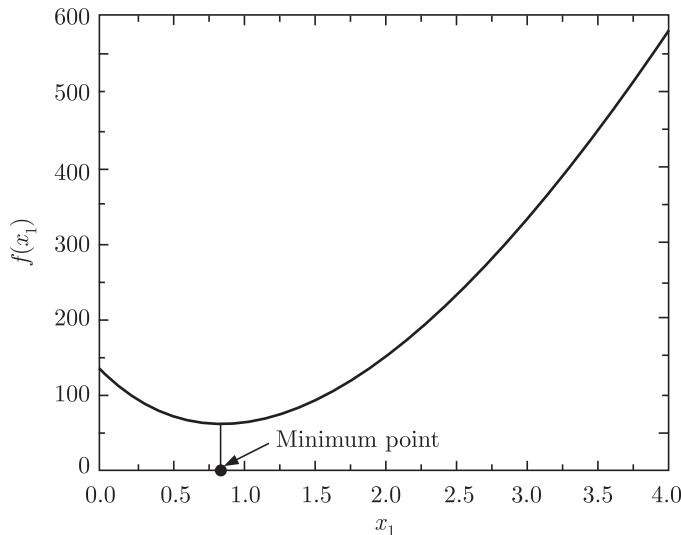
In the explicit method, the expression for the dependent variable in terms of the independent variable is directly substituted to the objective function. In the above problem, we may write the expression from the constraint as follows:

$$x_2 = \sqrt{26 - (x_1 - 5)^2}.$$

We also observe that when  $x_2$  is eliminated, the variable bounds on the independent variable  $x_1$  need to be changed to  $0 \leq x_1 \leq 4$ . Because, any value of  $4 < x_1 \leq 5$  will produce a value of  $x_2 > 5$ , which is not acceptable. Thus, the unconstrained minimization problem becomes

$$\begin{aligned} \text{Minimize } f(x_1) &= (x_1^2 + \sqrt{26 - (x_1 - 5)^2} - 11)^2 \\ &\quad + (x_1 + 26 - (x_1 - 5)^2 - 7)^2, \quad 0 \leq x_1 \leq 4. \end{aligned}$$

This function is plotted in Figure 4.11. Since the above problem is a single-variable minimization problem, we use the golden section search method described in Chapter 2 and obtain the minimum point in the interval  $(0, 4)$ :  $x_1^* = 0.829$ . The corresponding solution of the exercise problem is  $x^* = (0.829, 2.933)^T$  with a function value equal to 60.373. The variable substitution method adopted here may not be possible in general. In those cases, an implicit method needs to be used.



**Figure 4.11** The function  $f(x_1)$  vs.  $x_1$ . The plot shows that the function is unimodal in the interval  $0 \leq x_1 \leq 4$ .

In the implicit method, the dependent variables are not directly substituted; rather, for a set of independent variables, equality constraints are solved using a root-finding method to compute the dependent variables numerically. To solve the above problem, we first choose an initial feasible

point corresponding to the independent variable. Let us say we choose  $\bar{x}^{(0)} = x_1^{(0)} = 4$ . Using this initial value, an unconstrained minimization method is used to solve the following problem:

$$\text{Minimize } (x_1^2 + x_2(x_1) - 11)^2 + (x_1 + (x_2(x_1))^2 - 7)^2, \quad (4.22)$$

where  $0 \leq x_1 \leq 4$ .

The above objective function is a function of  $x_1$  only. The variable  $x_2$  is written as a function of  $x_1$ . Let us use the Newton-Raphson method (for more than one variable, a multivariable optimization technique may be used), which requires calculation of both first and second-order derivatives. We compute the derivatives numerically. The expression  $x_2(x_1)$  is computed by solving the following minimization problem (root-finding problem) for a given value of  $x_1 = x_1^{(t)}$ :

$$\text{Minimize } \text{abs}[26 - (x_1^{(t)} - 5)^2 - x_2^2], \quad 0 \leq x_2 \leq 5. \quad (4.23)$$

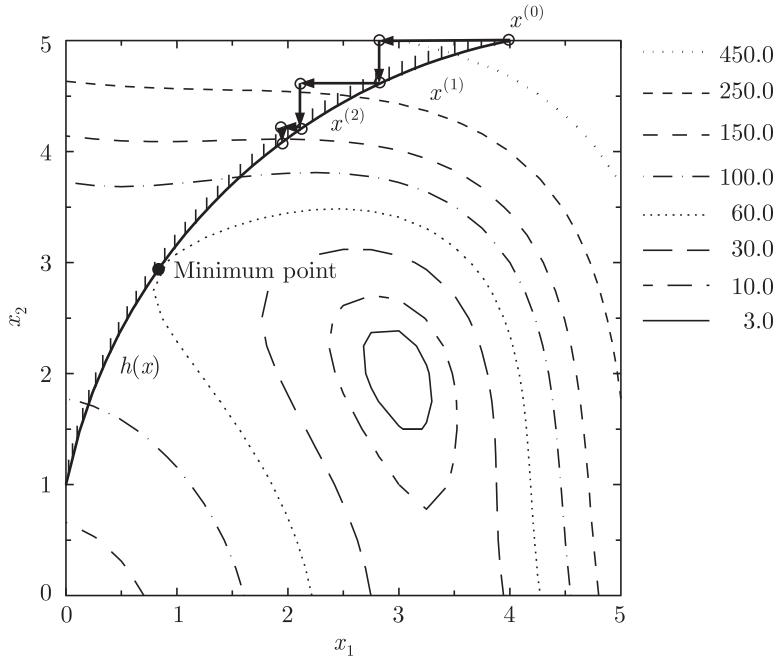
This minimization solves the equation  $26 - (x_1^{(t)} - 5)^2 - x_2^2 = 0$  for a fixed value of  $x_1^{(t)}$ . For example, at the initial point, the value of the dependent variable  $\hat{x} = x_2$  is found by solving the following problem:

$$\text{Minimize } \text{abs}[h(\bar{x}^{(0)}, \hat{x})] = \text{abs}[26 - (4 - 5)^2 - x_2^2], \quad 0 \leq x_2 \leq 5.$$

The solution obtained using the golden section search is  $\hat{x}^{(0)} = x_2^{(0)} = 5.00$ . For problems with more than one equality constraints, a multivariable minimization technique needs to be used. After  $x_2^{(0)}$  is found, the first iteration of the Newton-Raphson method applied to problem stated in (4.22) is

$$x_1(1) = x_1^{(0)} - \frac{f'(x_1^{(0)})}{f''(x_1^{(0)})},$$

where  $f(x_1) = (x_1^2 + x_2^{(0)} - 11)^2 + (x_1 + x_2^{(0)})^2 - 7^2$ . Computing the above equation results:  $x_1^{(1)} = 2.80$ . The resulting point is  $(2.80, 5.00)^T$ . As shown in Figure 4.12, this point is not a feasible point. Thus, in order to make the point feasible, we keep the same value of the independent variable  $x_1$ , but alter the dependent variable  $x_2$  so that the constraint is satisfied. This involves another optimization procedure (problem stated in (4.23)). The corresponding value for  $x_2$  found using the golden section search on the problem given in Equation (4.23) for  $x_1^{(1)} = 2.80$  is  $x_2^{(1)} = 4.60$ . The next iteration of the Newton-Raphson method results in the point  $(2.09, 4.60)^T$ . After adjusting this solution for constraint satisfaction, we find the point  $(2.09, 4.19)^T$ . Another iteration of Newton-Raphson search and constraint satisfaction using the golden section search yields the point  $x^{(3)} = (1.91, 4.06)^T$ . This process continues until the minimum point is found. Figure 4.12 shows



**Figure 4.12** The constraint  $h(x)$  and the progress of the implicit variable elimination method shown on a contour plot of the Himmelblau function.

how the implicit variable elimination method begins from the initial point  $x^{(0)} = (4, 5)^T$  and moves towards the true minimum point.

Thus, the implicit variable elimination method works by using two optimization problems—a unconstrained optimization problem followed by a root finding problem for constraint satisfaction. Besides, using the two optimization problems successively as illustrated here, the unconstrained optimization problem (given in Equation (4.22)) can be solved for  $\bar{x}$  ( $x_1$  here) and whenever an objective function value is required to be evaluated, the corresponding  $\hat{x}(\bar{x})$  ( $x_2$  here) can be found using a root-finding technique. The latter technique is discussed in detail in Section 4.9.

#### 4.5.2 Complex Search Method

The complex search method is similar to the simplex method of unconstrained search except that the constraints are handled in the former method. This method was developed by M. J. Box in 1965. The algorithm begins with a number of feasible points created at random. If a point is found to be infeasible, a new point is created using the previously generated feasible points. Usually, the infeasible point is *pushed* towards the centroid of the previously found feasible points. Once a set of feasible points is found, the worst point is reflected about the centroid of rest of the points to find a

new point. Depending on the feasibility and function value of the new point, the point is further modified or accepted. If the new point falls outside the variable boundaries, the point is modified to fall on the violated boundary. If the new point is infeasible, the point is retracted towards the feasible points. The worst point in the simplex is replaced by this new feasible point and the algorithm continues for the next iteration.

### Algorithm

**Step 1** Assume a bound in  $x$  ( $x^{(L)}$ ,  $x^{(U)}$ ), a reflection parameter  $\alpha$  and termination parameters  $\epsilon$ ,  $\delta$ .

**Step 2** Generate an initial set of  $P$  (usually  $2N$ ) feasible points. For each point

- (a) Sample  $N$  times to determine the point  $x_i^{(p)}$  in the given bound.
- (b) If  $x^{(p)}$  is infeasible, calculate  $\bar{x}$  (centroid) of current set of points and reset

$$x^{(p)} = x^{(p)} + \frac{1}{2}(\bar{x} - x^{(p)})$$

until  $x^{(p)}$  is feasible;

Else if  $x^{(p)}$  is feasible, continue with (a) until  $P$  points are created.

- (c) Evaluate  $f(x^{(p)})$  for  $p = 0, 1, 2, \dots, (P - 1)$ .

**Step 3** Carry out the reflection step:

- (a) Select  $x^R$  such that

$$f(x^R) = \max f(x^{(p)}) = F_{\max}.$$

- (b) Calculate the centroid  $\bar{x}$  (of points except  $x^R$ ) and the new point

$$x^m = \bar{x} + \alpha(\bar{x} - x^R).$$

(c) If  $x^m$  is feasible and  $f(x^m) \geq F_{\max}$ , retract half the distance to the centroid  $\bar{x}$ . Continue until  $f(x^m) < F_{\max}$ ;

Else if  $x^m$  is feasible and  $f(x^m) < F_{\max}$ , go to Step 5.

Else if  $x^m$  is infeasible, go to Step 4.

**Step 4** Check for feasibility of the solution

- (a) For all  $i$ , reset violated variable bounds:

$$\begin{aligned} \text{If } x_i^m < x_i^{(L)} \text{ set } x_i^m &= x_i^{(L)}. \\ \text{If } x_i^m > x_i^{(U)} \text{ set } x_i^m &= x_i^{(U)}. \end{aligned}$$

- (b) If the resulting  $x^m$  is infeasible, retract half the distance to the centroid. Continue until  $x^m$  is feasible. Go to Step 3(c).

**Step 5** Replace  $x^R$  by  $x^m$ . Check for termination.

(a) Calculate  $\bar{f} = \frac{1}{P} \sum_p f(x^{(p)})$  and  $\bar{x} = \frac{1}{P} \sum_p x^{(p)}$ .

(b) If  $\sqrt{\sum_p (f(x^{(p)}) - \bar{f})^2} \leq \epsilon$  and  $\sqrt{\sum_p \|x^{(p)} - \bar{x}\|^2} \leq \delta$

**Terminate:**

Else set  $k = k + 1$  and go to Step 3(a).

For the successful working of this algorithm, it is necessary that the feasible region should be convex; otherwise retraction of centroid point in Steps 3(c) and 4(b) may result in infeasible points. Often, the points of the simplex may end up close to a constraint boundary, in which case the speed of the algorithm tends to become somewhat slow. In problems where the feasible search space is narrow or the optimum lies at the constraint boundary, the algorithm is not very efficient. On the other hand, for problems with convex feasible search space and with the optimum well inside the search space, this algorithm is efficient.

### EXERCISE 4.5.2

Consider the constrained Himmelblau function in the range  $0 \leq x_1, x_2 \leq 5$ .

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

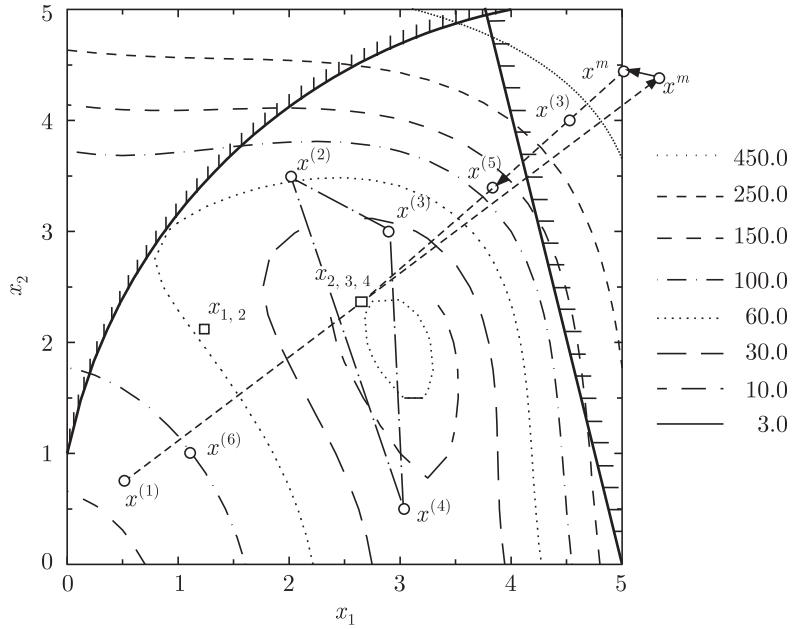
$$x_1, x_2 \geq 0.$$

**Step 1** The minimum and maximum bounds on both variables are taken to be 0 and 5, respectively. The reflection parameter  $\alpha = 1.3$  and convergence parameters  $\epsilon = \delta = 10^{-3}$  are chosen.

**Step 2** Since  $N = 2$ , we need to create a set of four feasible points. Since the points are required to be random in the chosen range, we create two random numbers to choose a point:

$$x_i = 0 + r_i(5 - 0).$$

(a) We create two random numbers  $r_1 = 0.10$  and  $r_2 = 0.15$ . Thus, the first point is  $x^{(1)} = (0.50, 0.75)^T$ . To investigate whether this point is feasible, we compute the constraints:  $g_1(x^{(1)}) = 5.19 > 0$  and  $g_2(x^{(1)}) = 17.25 > 0$ . Thus, the point  $x^{(1)}$  is feasible. The feasibility of the point can also be observed from Figure 4.13.



**Figure 4.13** Two iterations of the complex search method.

(a) We create two new random numbers to find the second point:  $r_1 = 0.4$  and  $r_2 = 0.7$ . Thus, the point is  $x^{(2)} = (2.00, 3.50)^T$ . Calculating the constraint values, we observe that the point is feasible (Figure 4.13).

(a) We create another set of random numbers:  $r_1 = 0.9$  and  $r_2 = 0.8$  to create the third point. The point is  $x^{(3)} = (4.50, 4.00)^T$ . For this point, we observe that  $g_1(x^{(3)}) = 9.75 > 0$ , but  $g_2(x^{(3)}) = -2.00 < 0$ , which is violated. Figure 4.13 shows that this point falls on the right-side of the constant  $g_2(x)$ . Thus the point is infeasible. In order to create a feasible point, we push the point towards the centroid of the first two points. This approach is likely to provide a feasible point.

(b) The centroid of the previous two points is

$$\bar{x} = (1.250, 2.125)^T,$$

which is marked  $x_{1,2}$  in the figure. The new point is

$$x^{(3)} = \begin{pmatrix} 4.500 \\ 4.000 \end{pmatrix} + \frac{1}{2} \left( \begin{pmatrix} 1.250 \\ 2.125 \end{pmatrix} - \begin{pmatrix} 4.500 \\ 4.000 \end{pmatrix} \right) = \begin{pmatrix} 2.875 \\ 3.062 \end{pmatrix}.$$

Geometrically, this point is obtained by pushing the point  $x^{(3)}$  halfway towards  $x_{1,2}$ . Calculating the constraint values we observe that the new point  $x^{(3)}$  is feasible. Thus, we accept the third point as  $x^{(3)} = (2.875, 3.062)^T$ .

(a) Another set of random numbers ( $r_1 = 0.6$ ,  $r_2 = 0.1$ ) creates a feasible point  $x^{(4)} = (3.0, 0.5)^T$ .

(c) The function values for these four points are  $f(x^{(1)}) = 135.25$ ,  $f(x^{(2)}) = 64.812$ ,  $f(x^{(3)}) = 27.711$ , and  $f(x^{(4)}) = 16.312$ .

**Step 3** After creating four feasible points, we now find out the worst point and reflect it over the centroid of rest of the points.

(a) The worst point is  $x^{(1)}$ , as can be seen from the above function values (or from the figure). Thus, we set  $x^R = x^{(1)}$  and  $F_{\max} = 135.250$ .

(b) The centroid of the second, the third, and the fourth points is  $\bar{x} = (2.625, 2.354)^T$  and is marked as  $x_{2,3,4}$  in the figure. The new point is computed by reflection as follows:

$$x^m = \begin{pmatrix} 2.625 \\ 2.354 \end{pmatrix} + 1.3 \left( \begin{pmatrix} 2.625 \\ 2.354 \end{pmatrix} - \begin{pmatrix} 0.500 \\ 0.750 \end{pmatrix} \right) = \begin{pmatrix} 5.387 \\ 4.439 \end{pmatrix}.$$

This point falls outside the feasible region as shown in the figure. By calculating the constraint values we observe that  $g_1(x^m) = 6.14$  and  $g_2(x^m) = -5.99$ , which is violated. Thus, the point  $x^m$  is infeasible. Since the point is infeasible, before we do any modification, we check whether the obtained point is close to a variable boundary or not.

**Step 4** We now check for the feasibility of the point  $x^m$ .

(a) Since  $x_1^m > x_1^{(U)} = 5$ , we set  $x_1^m = 5.0$ . Thus the new point is  $x^m = (5.000, 4.439)^T$ , which lies on the boundary  $x_1 = 5$ . This point is also infeasible; thus we move to Step 4(b).

(b) We retract half the distance to the centroid. This can be achieved by taking the average of the points  $\bar{x}$  ( $x_{2,3,4}$  in the figure) and  $x^m$ :

$$x^m = (\bar{x} + x^m)/2 = (3.813, 3.397)^T.$$

At this point, the constraints are not violated, thus the point is feasible. We accept this point and move to Step 3(c). This point is marked  $x^{(5)}$  in the figure.

**Step 3(c)** The function value at this point is  $f(x^{(5)}) = 117.874$ , which is smaller than  $F_{\max} = 135.250$ . We continue with Step 5.

**Step 5** We now form the new simplex and check for termination.

(a) We form the new simplex by replacing  $x^{(1)}$  by  $x^m$ . Computing the average function value of the simplex we obtain:  $\bar{f} = 61.02$  and  $\bar{x} = (2.094, 1.953)^T$ .

(b) The sum of the squared differences in function values from  $\bar{f}$  is found to be 8685.9 which is large compared to  $\epsilon^2$ . The quantity  $\sum_p \|x^{(p)} - \bar{x}\|^2$  is also large compared to  $\delta^2$ . This completes one iteration of the complex search algorithm. Figure 4.13 shows the progress of one iteration of this algorithm. In order to proceed to the next iteration, we need to move to Step 3(a), but we do not show the results here for brevity.

After another iteration of the complex search algorithm, the new point is found to be  $x^{(6)} = (1.081, 0.998)^T$  with a function value equal to  $f(x^{(6)}) = 102.264$ . The new simplex constitutes points  $x^{(2)}$ ,  $x^{(3)}$ ,  $x^{(4)}$ , and  $x^{(6)}$ . This process continues until the simplex size becomes smaller than the chosen termination parameters. The parameters  $P$  and  $\alpha$  are two important parameters for the successful working of the complex search method. The larger the value of  $P$  (number of points in the simplex), the better the convergence characteristics. But a large number of points in the simplex requires more function evaluations in each iteration. A compromise of  $P \approx 2N$  is usually followed (Box, 1965). As seen in the above exercise, a large value of  $\alpha$  may create points outside the feasible range. Although the infeasible points can be brought back to the feasible region by successive retraction, frequent retraction may cause the points to lie along the constraint boundaries. This reduces the search power considerably. Thus, the algorithm works well when the simplex points lie well inside the feasible region.

#### 4.5.3 Random Search Methods

Like the complex search method, the random search methods also work with a population of points. But instead of replacing the next point in the population by a point created in a structured manner, points are created either at random or by performing a unidirectional search along the random search directions. Here, we describe one such method. Since there is no specific search direction used in this method, random search methods work equally efficiently to many problems.

In the Luus and Jaakola method (1973), an initial point and an initial interval are chosen at random. Depending on the function values at a number of random points in the interval, the search interval is reduced at every iteration by a constant factor. The reliability of the algorithm increases as the number of sample points is increased. In the following algorithm,  $P$  points are considered at each iteration and  $Q$  such iterations are performed. Thus, if the initial interval in one variable is  $d_0$  and at every iteration the interval is reduced by a factor  $\epsilon$ , the final accuracy in the solution in that variable becomes  $(1 - \epsilon)^Q d_0$  and the required number of function evaluations is  $P \times Q$ .

#### Algorithm

**Step 1** Given an initial feasible point  $x^0$ , an initial range  $z^0$  such that the minimum,  $x^*$ , lies in  $(x^0 - \frac{1}{2}z^0, x^0 + \frac{1}{2}z^0)$ . Choose a parameter  $0 < \epsilon < 1$ . For each of  $Q$  blocks, initially set  $q = 1$  and  $p = 1$ .

**Step 2** For  $i = 1, 2, \dots, N$ , create points using a uniform distribution of  $r$  in the range  $(-0.5, 0.5)$ . Set  $x_i^{(p)} = x_i^{q-1} + rz_i^{q-1}$ .

**Step 3** If  $x^{(p)}$  is infeasible and  $p < P$ , repeat Step 2. If  $x^{(p)}$  is feasible, save  $x^{(p)}$  and  $f(x^{(p)})$ , increment  $p$  and repeat Step 2;

Else if  $p = P$ , set  $x^q$  to be the point that has the lowest  $f(x^{(p)})$  over all feasible  $x^{(p)}$  including  $x^{q-1}$  and reset  $p = 1$ .

**Step 4** Reduce the range via  $z_i^q = (1 - \epsilon)z_i^{q-1}$ .

**Step 5** If  $q > Q$ , **Terminate**;

Else increment  $q$  and continue with Step 2.

The suggested values of parameters are  $\epsilon = 0.05$ ,  $P = 100$ , and  $Q$  is related to the desired accuracy in the solution. It is to be noted that the obtained solution is not guaranteed to be the true optimum.

### EXERCISE 4.5.3

Consider again the constrained Himmelblau function:

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

$$x_1, x_2 \geq 0.$$

**Step 1** Let us assume that the initial point is  $x^0 = (3, 3)^T$ , the initial interval  $z^0 = (6, 6)^T$ . Other parameters are  $P = 3$  (in practice, a large value is suggested),  $Q = 10$ , and  $\epsilon = 0.25$  (in practice, a much smaller value is suggested). We set counters  $p = q = 1$ .

**Step 2** We create two random numbers between  $-0.5$  to  $0.5$ :  $r_1 = 0.018$  and  $r_2 = -0.260$ . The corresponding point is

$$x^{(1)} = \begin{pmatrix} 3 + (0.018)6 \\ 3 + (-0.260)6 \end{pmatrix} = \begin{pmatrix} 3.108 \\ 1.440 \end{pmatrix}.$$

**Step 3** The point  $x^{(1)}$  is feasible (checked by calculating constraint violations) and the function value at this point is  $f(x^{(1)}) = 3.338$ . Since  $p = 1 < P = 3$ , we create the second point.

**Step 2** Two more numbers in the interval  $(-0.5, 0.5)$  created at random are  $r_1 = -0.041$  and  $r_2 = -0.194$ . Thus, the second point is  $x^{(2)} = (2.754, 1.836)^T$ .

**Step 3** The point  $x^{(2)}$  is also feasible and the function value is  $f(x^{(2)}) = 3.299$ .

**Step 2** To create the third point, we create two new random numbers:  $r_1 = -0.464$  and  $r_2 = 0.149$ . The corresponding point is  $x^{(3)} = (0.216, 3.894)^T$ .

**Step 3** This point is not feasible, since the first constraint is violated ( $g_1(x^{(3)}) = -12.050$ ). This point is also an infeasible point, as shown in Figure 4.14. Thus, we do not accept this point, rather create a new point.

**Step 2** We choose two new random numbers  $r_1 = -0.344$  and  $r_2 = -0.405$ . The new point is

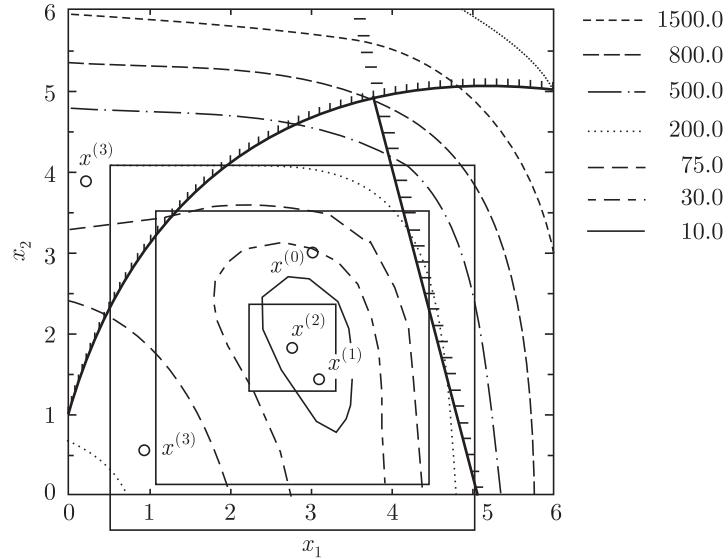
$$x^{(3)} = (0.936, 0.570)^T.$$

**Step 2** Since the point  $x^{(3)}$  is feasible, we compute the function value  $f(x^{(3)}) = 124.21$ . At this stage,  $p = 3 = P$ ; we find the best of all  $P$  feasible points. We observe that the best point is  $x^{(2)}$  with function value 3.299. Thus,  $x^1 = (2.754, 1.836)^T$ . We reset the counter  $p = 1$ .

**Step 4** We reduce the interval for the next iteration:

$$z^1 = (1 - 0.25)z^0 = (4.5, 4.5)^T.$$

**Step 5** Since  $q = 1 \not> Q = 10$ , we continue with Step 2. This completes one iteration of the Luus and Jaakola algorithm. Three points and the reduced search space are shown in Figure 4.14.



**Figure 4.14** Two iterations of random search method. The search region after five more iterations is also shown.

**Step 2** Creating two more random numbers in the interval  $(-0.5, 0.5)$  ( $r_1 = -0.284$  and  $r_2 = 0.311$ ), we obtain the first point for the second iteration:

$$x^{(1)} = \begin{pmatrix} 2.754 + (-0.284)4.5 \\ 1.836 + (0.311)4.5 \end{pmatrix} = \begin{pmatrix} 1.476 \\ 3.236 \end{pmatrix}.$$

**Step 3** The point  $x^{(1)}$  is feasible and has a function value  $f(x^{(1)}) = 56.030$ . We create two other points for comparison. The points are  $x^{(2)} = (3.857, 0.647)^T$  and  $x^{(3)} = (4.070, 2.490)^T$  with function values  $f(x^{(2)}) = 27.884$  and  $f(x^{(3)}) = 75.575$ , respectively. Comparing these points and the previously found best point  $x^1$ , we find that the current best point is  $x^2 = x^1 = (2.754, 1.836)^T$ .

**Step 4** We reduce the interval  $z^2 = 0.75z^1 = (3.375, 3.375)^T$ .

Thus, the search interval is centred around the point  $x^2$  with a reduced size. This process continues until the counter  $q$  is equal to the specified  $Q$ . The progress of the algorithm is shown in Figure 4.14. If the current best solution remains the best point for another five iterations, the search region (the small box in the vicinity of the true minimum) is reduced to a small size as shown in the figure. Since the search region is small, the probability of finding either the true minimum or a point close to the true optimum is high. The reduction of the search space from one iteration to another must be marginal; otherwise the algorithm may exclude the true optimum and converge to a wrong solution.

In case the feasible search space is very narrow, this method may be inefficient, creating many infeasible points. Instead of using this method in actually finding the optimum point, this method is usually used to find a feasible initial guess for other more sophisticated constrained optimization methods.

## 4.6 Linearized Search Techniques

In the linearized search techniques, the objective function as well as the constraints are linearized at a specified point. A linear programming (LP) method is used to solve the problem and to find a new point. Different LP methodologies are discussed in detail in the Appendix. The objective function and constraints are further linearized at the new point. This process continues until the algorithm converges close to the desired optimum point.

A nonlinear function  $f(x)$  can be linearized at a point  $x^{(0)}$  by considering the first-order term in the Taylor's series expansion of the function at that point:

$$f(x) = f(x^{(0)}) + \nabla f(x^{(0)})(x - x^{(0)}). \quad (4.24)$$

It is noteworthy that if the function  $f(x)$  is a linear function, the above first-order approximation is exact. For nonlinear functions, the above

approximation is valid close to the chosen point  $x^{(0)}$ . Two linearized search techniques are discussed here. One uses an LP algorithm successively to solve LP problems created by linearizing the objective function and all constraints at intermediate points. This method is known as the *Frank-Wolfe* algorithm. The other method also uses a successive LP algorithm but each time an LP problem is created by linearizing the objective function and only a few governing constraints. This method is known as the *cutting plane* method. These methods use gradient information of the constraints and the objective function.

In linearized search methods and in a number of other NLP methods, LP methodology is extensively used mainly due to its simplicity and availability of computer codes for implementing LP techniques. The LP methods are discussed in details in texts on operations research (Taha, 1989). Without deviating from the main focus of this book, we present a brief discussion on one linear programming technique in the Appendix.

#### 4.6.1 Frank-Wolfe Method

In the Frank-Wolfe method, all constraints and the objective function are first linearized at an initial point to form an LP problem. The simplex method of LP technique is used to find a new point by solving the LP problem. Thereafter, a unidirectional search is performed from the previous point to the new point. At the best point found using the unidirectional search, another LP problem is formed by linearizing the constraints and the objective function. The new LP problem is solved to find another new point. This procedure is repeated until a termination criterion is satisfied. The algorithm works well for NLP problems with convex feasible search space. Since LP problems are comparatively easier to solve, many NLP problems are solved by using this technique.

##### Algorithm

**Step 1** Assume an initial point  $x^{(0)}$ , two convergence parameters  $\epsilon$  and  $\delta$ . Set an iteration counter  $t = 0$ .

**Step 2** Calculate  $\nabla f(x^{(t)})$ . If  $\|\nabla f(x^{(t)})\| \leq \epsilon$ , **Terminate**;

Else go to Step 3.

**Step 3** Solve the following LP problem:

$$\text{Minimize } f(x^{(t)}) + \nabla f(x^{(t)})(x - x^{(t)})$$

subject to

$$g_j(x^{(t)}) + \nabla g_j(x^{(t)})(x - x^{(t)}) \geq 0, \quad j = 1, 2, \dots, J;$$

$$h_k(x^{(t)}) + \nabla h_k(x^{(t)})(x - x^{(t)}) = 0, \quad k = 1, 2, \dots, K;$$

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \dots, N.$$

Let  $y^{(t)}$  be the optimal solution to the above LP problem.

**Step 4** Find  $\alpha^{(t)}$  that minimizes  $f(x^{(t)} + \alpha(y^{(t)} - x^{(t)}))$  in the range  $\alpha \in (0, 1)$ .

**Step 5** Calculate  $x^{(t+1)} = x^{(t)} + \alpha^{(t)}(y^{(t)} - x^{(t)})$ .

**Step 6** If  $\|x^{(t+1)} - x^{(t)}\| < \delta \|x^{(t)}\|$  and if  $\|f(x^{(t+1)}) - f(x^{(t)})\| < \epsilon \|f(x^{(t)})\|$ , **Terminate**;

Else  $t = t + 1$  and go to Step 2.

As given in the above algorithm, the simulation can terminate either from Step 2 or from Step 6. But in most cases, the simulation terminates from Step 6. The above algorithm works similar to the steepest descent search method. The convergence of this algorithm to a Kuhn-Tucker point is proved by Zangwill (1969). Since linear approximations of the objective function and the constraints are used, in highly nonlinear problems the search process may have to be restricted to a small neighbourhood of the point  $x^{(t)}$ .

#### EXERCISE 4.6.1

Consider again the Himmelblau function:

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

$$x_1, x_2 \geq 0.$$

**Step 1** We choose an initial point  $x^{(0)} = (0, 0)^T$  and convergence parameters  $\epsilon = \delta = 10^{-3}$ . We set the iteration counter  $t = 0$ .

**Step 2** The gradient at this point is  $\nabla f(x^{(0)}) = (-14, -22)^T$ . Since the magnitude of the gradient is not close to zero, we move to Step 3.

**Step 3** At this step, we first form the LP problem by linearizing the objective function and all constraints with respect to  $x = (x_1, x_2)^T$ . In the above problem, the linearized objective function is calculated as follows:

$$\begin{aligned} f(x) &= f(x^{(0)}) + \nabla f(x^{(0)})(x - x^{(0)}), \\ &= 170 + (-14, -22) \begin{pmatrix} x_1 - 0 \\ x_2 - 0 \end{pmatrix}, \\ &= -14x_1 - 22x_2 + 170. \end{aligned}$$

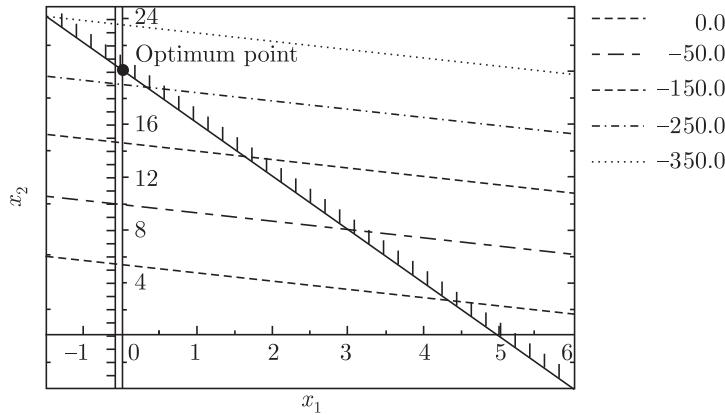
This way the constraints can also be linearized. The complete LP problem is given as follows:

$$\text{Minimize } -14x_1 - 22x_2 + 170$$

subject to

$$\begin{aligned} 10x_1 + 1 &\geq 0, \\ -4x_1 - x_2 + 20 &\geq 0, \\ x_1, x_2 &\geq 0. \end{aligned}$$

The objective function and the two constraints are linear functions of  $x_1$  and  $x_2$ . Also observe that the linear constraint  $g_2(x)$  remains the same after linearization. The simplex method of LP search can now be used to solve the above problem. Since there are only two variables, we solve the above LP problem graphically. Interested readers may refer to the Appendix for the simplex technique of LP method. Figure 4.15 shows the feasible region of above LP problem. By plotting the contours of the objective function at various function values, we observe that the solution to the above LP problem is  $y^{(0)} = (0, 20)^T$ .



**Figure 4.15** The linear programming problem in the first iteration of the Frank-Wolfe algorithm. The optimum point is  $(0, 20)^T$ .

**Step 4** The search direction for the unidirectional search is  $y^{(0)} - x^{(0)} = (0, 20)^T$ . Performing the golden section search in the domain  $(0 \leq \alpha \leq 1)$  yields the minimum  $\alpha^{(0)} = 0.145$ .

**Step 5** Thus, the new point is  $x^{(1)} = (0.000, 2.898)^T$ .

**Step 6** Since points  $x^{(0)}$  and  $x^{(1)}$  are not close enough (with respect to  $\epsilon$ ), we continue with Step 2. This completes the first iteration of the Frank-Wolfe algorithm.

**Step 2** The gradient at the new point is  $\nabla f(x^{(1)}) = (2.797, -0.016)^T$ .

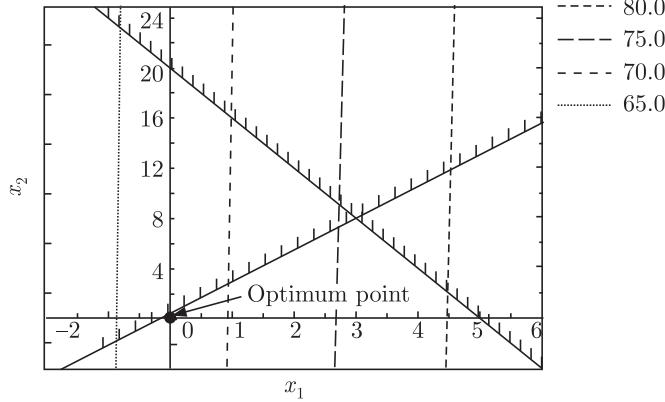
**Step 3** We form another LP problem by linearizing the objective function and the two constraints at this point.

$$\text{Minimize } 2.797x_1 - 0.016x_2 + 67.55$$

subject to

$$\begin{aligned} 10x_1 - 3.885x_2 + 1 &\geq 0, \\ -4x_1 - x_2 + 20 &\geq 0, \\ x_1, x_2 &\geq 0. \end{aligned}$$

The solution to this LP is  $y^{(1)} = (0, 0.257)^T$ , as shown graphically in Figure 4.16. The usual tableau method of simplex search (see Appendix) can also be used to arrive at this solution.



**Figure 4.16** The linear programming problem in the second iteration of the Frank-Wolfe algorithm. The optimum solution is  $(0, 0.257)^T$ .

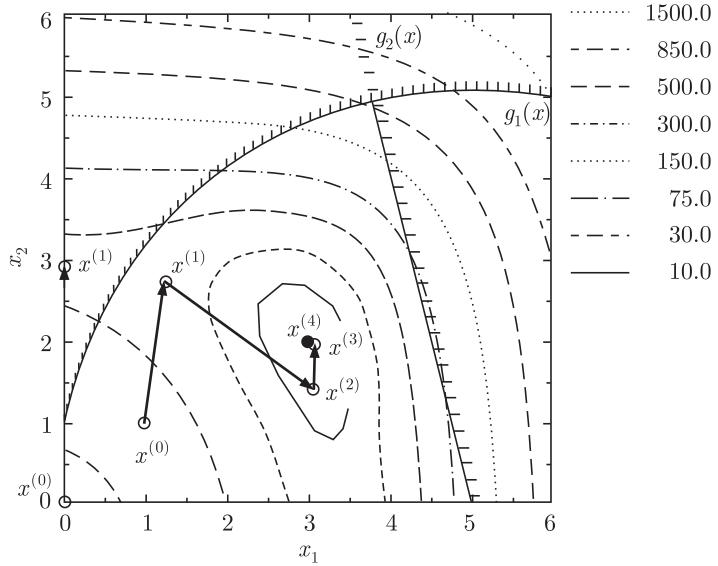
**Step 4** By performing another unidirectional search along  $(y^{(1)} - x^{(1)})$ , we obtain  $\alpha^{(1)} = 0$ .

**Step 5** Thus, the new point is  $x^{(2)} = (0, 2.898)^T$ , which is identical to the previous point  $x^{(1)}$ .

**Step 6** Since the points  $x^{(1)}$  and  $x^{(2)}$  are identical, the algorithm prematurely terminates to a wrong solution. The history of the intermediate points is shown on a contour plot of the objective function in Figure 4.17, where starting from the point  $(0, 0)^T$  the algorithm converges to the point  $(0, 2.898)^T$  on the  $x_2$  axis. This premature convergence to a wrong solution is one of the drawbacks of the Frank-Wolfe algorithm. When this happens, the algorithm is usually restarted from a different initial point.

**Step 1** We restart the algorithm from an initial point  $x^{(0)} = (1, 1)^T$ . Other parameters are the same as before.

**Step 2** The gradient at this point (calculated numerically) is  $\nabla f(x^{(1)}) = (-46, -38)^T$ . Since the gradient is not small, we proceed to Step 3.



**Figure 4.17** Intermediate points obtained using the Frank-Wolfe algorithm. The first attempt with an initial solution  $x^{(0)} = (0,0)^T$  prematurely converges to a wrong solution. A restart with an initial solution  $x^{(0)} = (1,1)^T$  finds the true optimum.

**Step 3** We form an LP problem at this point by linearizing all constraints and the objective function:

$$\text{Minimize} \quad -46x_1 - 38x_2 + 190$$

subject to

$$\begin{aligned} 8x_1 - 2x_2 + 3 &\geq 0, \\ -4x_1 - x_2 + 20 &\geq 0, \\ x_1, x_2 &\geq 0. \end{aligned}$$

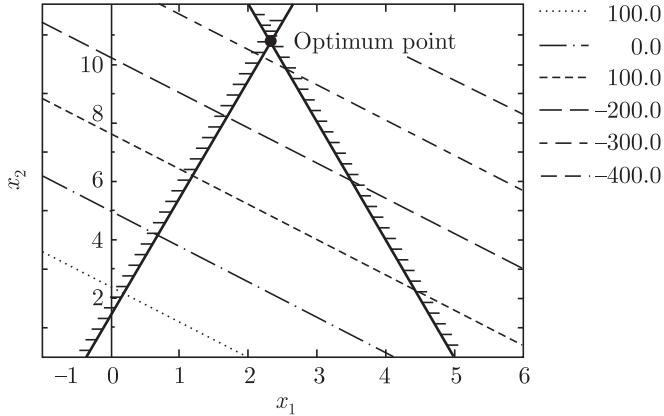
By solving the above LP problem graphically (Figure 4.18), we obtain the solution  $y^{(0)} = (2.312, 10.750)^T$ .

**Step 4** The search direction is  $(y^{(0)} - x^{(0)}) = (1.312, 9.750)^T$ . Performing the golden section search along this direction, we obtain the minimum  $\alpha^{(0)} = 0.176$ .

**Step 5** The minimum point is

$$x^{(1)} = x^{(0)} + 0.176(y^{(0)} - x^{(0)}) = (1.231, 2.718)^T.$$

**Step 6** Since the points  $x^{(0)}$  and  $x^{(1)}$  are not close to each other, we move to Step 2. This is the end of one iteration of the Frank-Wolfe algorithm with the new starting point.



**Figure 4.18** The linear programming problem in the first iteration of the Frank-Wolfe algorithm. The algorithm is started from the point  $(1, 1)^T$ . The optimum solution is  $(2.312, 10.750)^T$ .

**Step 2** The gradient of the objective function at the new point is

$$\nabla f(x^{(1)}) = (-30.082, 4.063)^T.$$

**Step 3** We form a new LP at the point  $x^{(1)}$ :

$$\text{Minimize} \quad -30.082x_1 + 4.063x_2 + 74.395$$

subject to

$$\begin{aligned} 7.538x_1 - 3.936x_2 + 5.830 &\geq 0, \\ -4x_1 - x_2 + 20 &\geq 0, \\ x_1, x_2 &\geq 0. \end{aligned}$$

The solution to this LP problem is found to be  $y^{(1)} = (5, 0)^T$ , as shown in Figure 4.19.

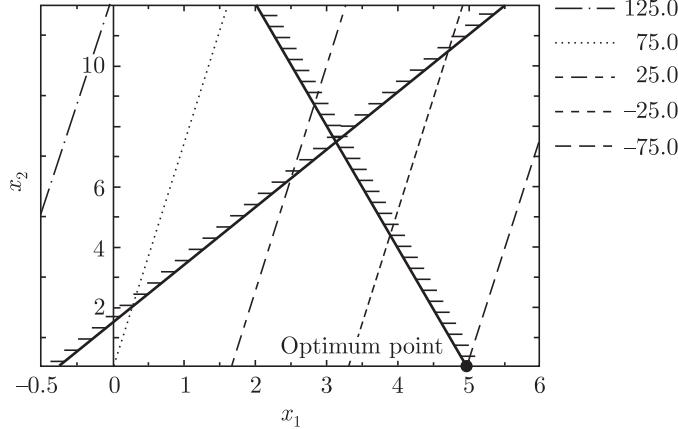
**Step 4** The search direction is  $(y^{(1)} - x^{(1)}) = (3.769, -2.718)^T$ . Performing the golden section search in the interval  $(0, 1)$ , we obtain the minimum  $\alpha^{(1)} = 0.478$ .

**Step 5** The corresponding point is  $x^{(2)} = (3.033, 1.418)^T$ .

**Step 6** Since points  $x^{(1)}$  and  $x^{(2)}$  are not close to each other, this procedure is continued until the algorithm converges to a point.

Two subsequent iterations of this algorithm result in points  $x^{(3)} = (3.076, 1.941)^T$  and  $x^{(4)} = (3.002, 1.973)^T$  with function values equal to 0.187 and 0.011, respectively.

The Frank-Wolfe algorithm restricts the unidirectional search in the interval  $(x^{(t)}, y^{(t)})$ . In order to make the algorithm more flexible, this



**Figure 4.19** The linear programming problem in the second iteration of the Frank-Wolfe algorithm. The optimum solution is  $(5, 0)^T$ .

restriction may be ignored. The bounding phase algorithm may be used to first bracket the minimum and then the golden section search may be used to find the minimum with a desired accuracy in that interval. As demonstrated in the above exercise, the Frank-Wolfe algorithm may sometimes converge to a wrong solution. In order to avoid this problem, a restart from a new random point is often necessary.

#### 4.6.2 Cutting Plane Method

The cutting plane method begins with a user-defined search space. At every iteration, some part of that search space is *cut* (or eliminated) by constructing linear hyperplanes from the most violated constraint at the current point. The objective function is minimized in the resulting search space and a new point is found. Depending on the obtained point, a certain portion of the search space is further eliminated. Although the algorithm is designed to work for linear objective functions and convex feasible regions, the algorithm has been applied successfully to nonlinear objective functions as well. Since the algorithm works most efficiently for linear objective functions and also the obtained cutting planes (or constraints) are always linear, a linear programming technique is used to solve every subproblem.

We first describe the cutting plane algorithm for solving NLP problems with linear objective functions of the following type:

$$\text{Minimize} \quad \sum_{i=1}^N c_i x_i$$

subject to

$$\begin{aligned} g_j(x) &\geq 0, & j = 1, 2, \dots, J; \\ x_i^{(L)} &\leq x_i \leq x_i^{(U)}, & i = 1, 2, \dots, N. \end{aligned}$$

We shall discuss a modified algorithm later to handle nonlinear objective functions, but it is noteworthy that the cutting plane algorithm works most efficiently for linear objective functions. The algorithm usually begins with a search space defined only by the variable bounds

$$Z^0 = \{x : x_i^{(L)} \leq x_i \leq x_i^{(U)}, i = 1, 2, \dots, N\},$$

such that  $Z^0$  contains the true feasible region. However, any other search space that includes the optimum point can also be chosen. At any iteration  $t$ , a new cutting plane ( $p^{(t)} \geq 0$ ) is found and a new search space  $Z^t$  is found by performing an intersection of the previous search space  $Z^{t-1}$  with the new cutting plane  $p^{(t)}$ .

### Algorithm

**Step 1** Choose a constraint tolerance  $\epsilon$  and an initial search space  $Z^0$ . Solve the following LP problem:

$$\text{Minimize} \quad \sum_{i=1}^N c_i x_i$$

subject to

$$x \in Z^0.$$

Let us say that the solution is  $x^{(1)}$ . Set a counter  $k = 1$ .

**Step 2** Find  $m$  such that

$$-g_m(x^{(k)}) = \max [-g_j(x^{(k)}), 0; j = 1, 2, \dots, J].$$

If  $g_m(x^{(k)}) > -\epsilon$ , **Terminate**;

Else go to Step 3.

**Step 3** Construct a cutting plane:

$$p^{(k)}(x) \equiv \tilde{g}_m(x; x^{(k)}) = g_m(x^{(k)}) + \nabla g_m(x^{(k)})(x - x^{(k)}) \geq 0.$$

Let  $H^{(k)}$  define the space  $H^{(k)} = \{x : p^{(k)}(x) \geq 0\}$ . Solve the following LP problem:

$$\text{Minimize} \quad \sum_{i=1}^N c_i x_i$$

subject to

$$x \in Z^{(k-1)} \cap H^{(k)}.$$

Designate the solution  $x^{(k+1)}$ .

**Step 4** Set  $Z^{(k)} = Z^{(k-1)} \cap H^{(k)}$ ,  $k = k + 1$ , and go to Step 2.

We illustrate the working of this algorithm by using a numerical exercise problem. The constrained Himmelblau function used in the previous exercise problems has a convex feasible region but the objective function is nonlinear. Thus, the above algorithm cannot be directly used to solve that problem directly. To illustrate the proceedings of the algorithm better, we keep the same feasible region but consider a linear objective function (linearized version of the Himmelblau function at the point  $(0, 0)^T$ ).

### EXERCISE 4.6.2

Consider the following NLP problem:

$$\text{Minimize } 170 - 14x_1 - 22x_2$$

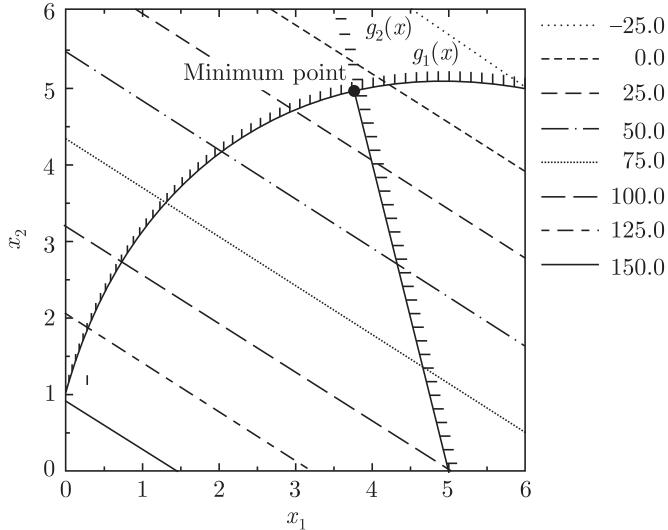
subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

$$x_1, x_2 \geq 0.$$

The optimum point of this problem is the intersection point of the constraints  $g_1(x)$  and  $g_2(x)$ :  $x^* = (3.763, 4.947)^T$  with a function value equal to  $f(x^*) = 8.481$ . The objective function, constraints, and the minimum point are shown in Figure 4.20.



**Figure 4.20** The NLP problem used to illustrate the basic cutting plane algorithm. The minimum point lies at  $x^* = (3.763, 4.947)^T$  with a function value equal to 8.481.

**Step 1** We choose a termination parameter  $\epsilon = 10^{-3}$  and an initial search space

$$Z^0 = \text{all } x \text{ satisfying } \begin{cases} 0 \leq x_1 \leq 6 \\ 0 \leq x_2 \leq 6 \end{cases}$$

Thus, the LP problem to be solved is as follows:

$$\text{Minimize } 170 - 14x_1 - 22x_2$$

subject to

$$0 \leq x_1 \leq 6, \quad 0 \leq x_2 \leq 6.$$

An LP programming technique can be used to solve this problem. Since the minimum of the problem is one of four corners of the search space  $((0, 0)^T, (6, 0)^T, (6, 6)^T, \text{ and } (0, 6)^T)$ , we solve the problem graphically. It is found that the solution to the above LP problem is  $x^{(1)} = (6, 6)^T$ .

**Step 2** At this step, we calculate the constraint violations at the point  $x^{(1)}$ . The corresponding constraint values are  $g_1(x^{(1)}) = -11$  and  $g_2(x^{(1)}) = -10$ . Thus, both constraints are violated at this point. Since, the maximum of negative of the constraint violations and zero (that is, the maximum of 11, 10, and 0) is 11 (which corresponds to the first constraint), the first constraint is violated maximally at this point. We set  $m = 1$ . Since,  $g_1(x^{(1)}) \not> -\epsilon$ , we go to Step 3.

**Step 3** At this step, we construct a cutting plane at  $x^{(1)}$  by linearizing the first constraint:

$$\begin{aligned} p^{(1)}(x) &\equiv g_1(x^{(1)}) + \nabla g_1(x^{(1)})(x - x^{(1)}) \geq 0, \\ &= -11 + (-2, -12)^T \begin{pmatrix} x_1 - 6 \\ x_2 - 6 \end{pmatrix}, \\ &= 73 - 2x_1 - 12x_2 \geq 0. \end{aligned}$$

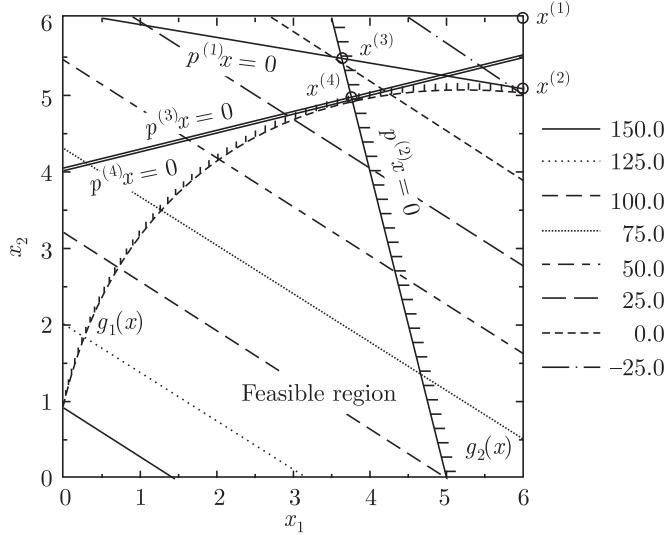
The vector  $\nabla g_1(x^{(1)})$  is computed numerically using the central difference technique described in Chapter 3. With the new constraint, we now construct the second LP problem by including a new constraint  $p^{(1)}(x) \geq 0$ :

$$\text{Minimize } 170 - 14x_1 - 22x_2$$

subject to

$$\begin{aligned} 73 - 2x_1 - 12x_2 &\geq 0, \\ 0 \leq x_1 &\leq 6, \\ 0 \leq x_2 &\leq 6. \end{aligned}$$

Since the new constraint added to the previous problem is linear, the resulting problem is an LP problem. The solution to this LP problem can be found graphically. We solve this problem in Figure 4.21. The obtained solution



**Figure 4.21** Four cutting planes and intermediate solutions. Note how the initial square region is cut by various cutting planes to take the shape of the true feasible region.

is  $x^{(2)} = (6.000, 5.083)^T$  having a function value equal to  $-25.826$ . As can be seen from the figure, this point is an infeasible point violating both the original constraints ( $g_1(x^{(2)}) = -0.834$  and  $g_2(x^{(2)}) = -9.083$ ).

**Step 4** The new search space is  $Z^1 = Z^0 \cap (p^{(1)}(x) \geq 0)$ , or

$$Z^1 = \text{all } x \text{ satisfying } \begin{cases} 0 \leq x_1 \leq 6 \\ 0 \leq x_2 \leq 6 \\ 73 - 2x_1 - 12x_2 \geq 0 \end{cases}$$

We increment the iteration counter to  $k = 2$  and proceed to Step 2. This completes one iteration of the cutting plane method. Note that the constraint  $p^{(1)}(x) \geq 0$  does not eliminate any portion of the feasible search space of the original problem.

**Step 2** At  $x^{(2)}$ , the constraint violations are  $g_1(x^{(2)}) = -0.837$  and  $g_2(x^{(2)}) = -9.083$ . Thus, the critical constraint is the second constraint and we set  $m = 2$ . Since the maximum constraint violation is not very small, we go to Step 3.

**Step 3** At the point  $x^{(2)}$ , the cutting plane obtained from the second constraint is calculated as follows:

$$\begin{aligned} p^{(2)}(x) &\equiv g_2(x^{(2)}) + \nabla g_2(x^{(2)})(x - x^{(2)}) \geq 0, \\ &\equiv 20 - 4x_1 - x_2 \geq 0. \end{aligned}$$

It is interesting to note that the constraint  $p^{(2)}(x) \geq 0$  is the same as the original constraint  $g_2(x) \geq 0$ . Since  $g_2(x)$  is linear, the linearization of this constraint at any point will always produce the same constraint. With the new constraint, we form another LP problem:

$$\text{Minimize } 170 - 14x_1 - 22x_2$$

subject to

$$20 - 4x_1 - x_2 \geq 0,$$

$$73 - 2x_1 - 12x_2 \geq 0,$$

$$0 \leq x_1 \leq 6,$$

$$0 \leq x_2 \leq 6.$$

The solution to this problem (as can be seen in Figure 4.21) is  $x^{(3)} = (3.630, 5.478)^T$ . The function value at this point is  $f(x^{(3)}) = -1.336$ . This point is also an infeasible point violating the constraint  $g_1(x)$  only.

**Step 4** The new and reduced search space is given by

$$Z^2 = \text{all } x \text{ satisfying } \left\{ \begin{array}{l} 0 \leq x_1 \leq 6 \\ 0 \leq x_2 \leq 6 \\ 73 - 2x_1 - 12x_2 \geq 0 \\ 20 - 4x_1 - x_2 \geq 0 \end{array} \right.$$

We increment the counter  $k$  to 3 and go to Step 2.

We continue this procedure until the termination criteria are met. The successive cutting planes and the feasible regions are shown in Figure 4.21. After two more iterations, the point  $x^{(4)} = (3.756, 4.974)^T$  is obtained. At this point, the function value is  $f(x^{(4)}) = 7.988$ , which is close to that of the true minimum point.

Figure 4.21 depicts how the cutting planes have eliminated most of the infeasible regions in the original large search space ( $Z^0$ ) and obtained a search space close to the true feasible search space. It is important to note that for linear objective functions the constrained minimum point always lies on a

constraint surface. Thus, after a number of iterations when the cutting planes surround the true feasible search space closely, the solution to the LP problem constructed by these cutting planes will be close to the true minimum point.

However, there are some disadvantages with this method. Firstly, the method can only be applied to convex feasible region and linear objective function efficiently. Secondly, the method cannot be terminated prematurely, because at any iteration the obtained solution is usually not feasible. Thirdly, the algorithm creates a new constraint at every iteration. Soon, the number of constraints becomes too many and the speed of the algorithm slows down. However, as seen from Figure 4.21, some constraints may dominate other constraints which may be eliminated from further consideration. In the above problem, the constraint  $p^{(3)}(x) \geq 0$  implies the constraint  $p^{(4)}(x) \geq 0$ . Thus, we can eliminate the former constraint from further consideration. This problem can be alleviated by advocating a *cut-deletion* procedure mentioned in the following algorithm.

In the cut-deletion method, at every iteration  $t$ , some constraints are deleted based on their inactiveness and the objective function value at the current point. For each previously created constraint, two conditions are checked. At first, the constraints are checked for inactiveness at the current point. Secondly, the objective function value at the current point is compared with the penalized function value at the point where each constraint is created. If the constraint is found to be inactive and the current function value is greater than the penalized function value at the previously found best point, there exists at least one other constraint that dominates this constraint at the point. Thus, this constraint is included in the set for deletion. This procedure requires the storage of all previously found best points corresponding to each constraint. The cut-deletion algorithm is obtained by replacing Step 4 of the cutting plane algorithm by two steps, given as follows:

### Algorithm

**Step 4(a)** Determine the set  $D$  of cutting planes to be deleted. For each  $i \in I^{(t)} \cup \{t\}$ , if both

$$p^{(i)}(x^{(t+1)}) > 0 (> \epsilon), \quad f(x^{(t+1)}) \geq f(x^{(i)}) + p^{(i)}(x^{(i)})$$

then include  $i$  in the set  $D$ . Let  $I^{(t+1)} = I^{(t)} \cup \{t\} - D$ .

**Step 4(b)** Set  $Z^{t+1} = Z^0 \cap \{x : p^{(i)}(x) \geq 0, i \in I^{(t+1)}\}$ ,  $t = t + 1$ , and go to Step 2.

We illustrate the working of this modified algorithm on the same problem considered earlier. At the third iteration ( $t = 3$ ), we have the following constraints:

$$\text{Minimize } 170 - 14x_1 - 22x_2$$

subject to

$$\begin{aligned} p^{(3)}(x) &= 44.22 + 2.74x_1 - 10.96x_2 \geq 0, \\ p^{(2)}(x) &= 20 - 4x_1 - x_2 \geq 0, \\ p^{(1)}(x) &= 73 - 2x_1 - 12x_2 \geq 0, \\ 0 \leq x_1, x_2 &\leq 6. \end{aligned}$$

At the end of Step 3, we have solved the LP problem and found the solution  $x^{(4)} = (3.756, 4.974)^T$ . The third constraint is also created at the current step. Thus, the set of constraints that are accumulated but not deleted before this iteration is  $I^{(t)} = \{1, 2\}$ . Since none of the constraints is deleted yet,  $D = \emptyset$ , an empty set. We now proceed to Step 4 to find out the constraints to be deleted.

**Step 4(a)** We consider constraints in the set  $I^{(t)} \cup \{t\}$  or  $\{1, 2, 3\}$ . For the first constraint,

$$\begin{aligned} p^{(1)}(x^{(4)}) &= 5.802 > 0, \\ f(x^{(4)}) &= 45.55 > f(x^{(1)}) + p^{(1)}(x^{(1)}) = -46 - 11 = -57. \end{aligned}$$

Thus, we include this constraint in the set  $D$ . We update the deletion set  $D = \{1\}$ . We now check the second constraint for deletion and find that  $p^{(2)}(x^{(4)}) \not> 0$ . Thus, we do not delete the second constraint. Similarly, we find that  $p^{(3)}(x^{(4)}) \not> 0$  and we do not delete it either. Before we move to Step 4(b), we update the set

$$I^{(4)} = \{1, 2, 3\} - \{1\} \equiv \{2, 3\}.$$

**Step 4(b)** At this step, we modify the constraint set:

$$Z^4 = \text{all } x \text{ satisfying } \left\{ \begin{array}{l} 0 \leq x_1 \leq 6 \\ 0 \leq x_2 \leq 6 \\ 20 - 4x_1 - x_2 \geq 0 \\ 44.22 + 2.74x_1 - 10.96x_2 \geq 0 \end{array} \right.$$

Note that the first constraint  $p^{(1)}(x) \geq 0$  has been eliminated from further consideration. As seen from Figure 4.21, the combination of the second constraint ( $p^{(2)}(x) \geq 0$ ), the third constraint ( $p^{(3)}(x) \geq 0$ ), and the variable bounds take care of the first constraint ( $p^{(1)}(x) \geq 0$ ) at point  $x^{(4)}$ . The cut-deletion method helps in reducing the number of constraints to be carried along, thereby reducing the effort in solving successive LP problems. Now, we set  $t = 4$  and move to Step 2. This completes the third iteration.

**Step 2** At this step, we find the most violated constraint first. It is found that the constraint  $g_1(x)$  is violated the most.

**Step 3** Thus, we form the new cutting plane:

$$p^{(4)}(x) = 39.89 + 2.48x_1 - 9.95x_2 \geq 0.$$

With this constraint, the LP problem becomes as follows:

$$\text{Minimize } 170 - 14x_1 - 22x_2$$

subject to

$$39.89 + 2.48x_1 - 9.95x_2 \geq 0,$$

$$44.22 + 2.74x_1 - 10.96x_2 \geq 0,$$

$$20 - 4x_1 - x_2 \geq 0,$$

$$0 \leq x_1, x_2 \leq 6.$$

Notice that the above problem contains the second, third, and fourth cutting planes; the first cutting plane is eliminated. The solution to this LP problem is  $x^{(5)} = (3.763, 4.948)^T$  with a function value  $f(x^{(5)}) = 8.462$ , which is now very close to the actual minimum of the problem. We now move to Steps 4(a) and 4(b) for deletion of constraints.

This process continues until the convergence criterion is met. With the cut-deletion procedure, only a few constraints are used in each LP problem.

The cutting plane method described above can also be used to solve convex, nonlinear objective functions with a simple modification. A new variable  $x_0$  is introduced with an additional artificial constraint:  $x_0 \geq f(x)$ . Thus, the new NLP problem becomes as follows:

$$\text{Minimize } x_0$$

subject to

$$g_j(x) \geq 0, \quad j = 1, 2, \dots, J;$$

$$x_0 - f(x) \geq 0,$$

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \dots, N.$$

The original objective of minimizing  $f(x)$  is achieved by minimizing a variable  $x_0$  and simultaneously satisfying a constraint  $x_0 \geq f(x)$ . This transformation makes the objective function always linear and independent of  $N$  design variables. Since  $f(x)$  is assumed to be convex, the extra constraint also makes the feasible region convex.

We illustrate the working of this modified problem by showing hand calculations of one or two iterations of the cutting plane method.

**EXERCISE 4.6.3**

Since the Himmelblau function is not convex, we cannot use the modified cutting plane method described above to solve the problem. To illustrate the working of the cutting plane method on nonlinear objective functions, we solve another problem with a convex objective function:

$$\text{Minimize } (x_1 - 3)^2 + (x_2 - 2)^2$$

subject to

$$26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$20 - 4x_1 - x_2 \geq 0,$$

$$x_1, x_2 \geq 0.$$

The minimum of this NLP problem lies at the point  $x^* = (3, 2)^T$ . Since the original function has two variables, the addition of the artificial variable  $x_0$  takes the total number of variables to three. Thus, the transformed problem becomes as follows:

$$\text{Minimize } x_0$$

subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

$$g_3(x) = x_0 - (x_1 - 3)^2 - (x_2 - 2)^2 \geq 0,$$

$$x_1, x_2 \geq 0.$$

In order to take care of the nonlinear objective function, an artificial constraint  $g_3(x)$  is added to the original problem. We choose an initial search space

$$Z^0 = \text{all } x \text{ satisfying } \begin{cases} 0 \leq x_1 \leq 6 \\ 0 \leq x_2 \leq 6 \\ 0 \leq x_0 \leq 30 \end{cases}$$

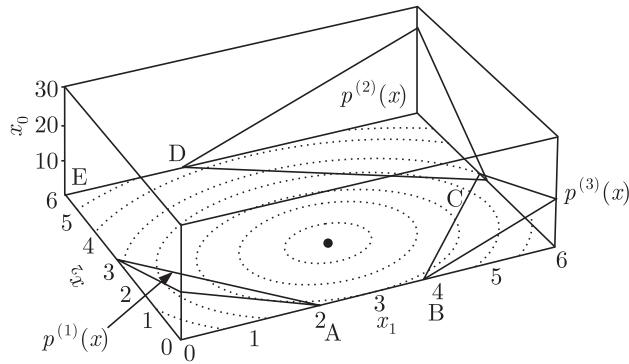
The constraints are plotted in a three-dimensional search space as shown in Figure 4.22. The minimum point is the solution  $x^* = (x_0^*, x_1^*, x_2^*) = (0, 3, 2)^T$ .

**Step 1** We formulate the first LP problem:

$$\text{Minimize } x_0$$

subject to

$$0 \leq x_1 \leq 6, \quad 0 \leq x_2 \leq 6, \quad 0 \leq x_0 \leq 30.$$



**Figure 4.22** The NLP problem and subsequent cutting planes.

We observe that any point on the plane  $x_0 = 0$  is a solution to the above problem. We choose the point  $x^{(1)} = (x_0^{(1)}, x_1^{(1)}, x_2^{(1)})^T = (0, 0, 0)^T$ .

**Step 2** We observe that the third constraint is maximally violated at this point. Thus,  $m = 3$ .

**Step 3** We construct a new cutting plane from the third constraint:  $p^{(1)}(x) = -13 + x_0 + 6x_1 + 4x_2 \geq 0$ . Thus, the new LP problem has this additional constraint. The feasible region is shown in Figure 4.22. We solve the new LP problem and observe again that there are infinite solutions. We choose the solution  $x^{(2)} = (0, 6, 6)^T$ .

**Step 4** The new search space is  $Z^1 = Z^0 \cap (p^{(1)} \geq 0)$ . We increment  $t$  to 2 and go to Step 2.

**Step 2** At this step, we observe that the third constraint is violated the most. Thus, we form the next cutting plane based on the third constraint.

**Step 3** The new cutting plane is  $p^{(2)}(x) = x_0 - 6x_1 - 8x_2 + 59 \geq 0$ . The new LP problem is formed by adding this constraint and the feasible region is shown in Figure 4.22. At this point, the LP problem is solved and a new cutting plane is found.

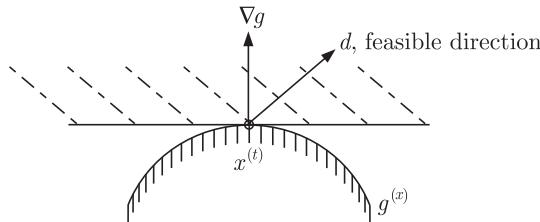
This way, cutting planes can be formed at various points and resulting feasible region narrows. Finally, the feasible region is small enough to find the optimal solution. Cut-deletion method can be used to keep the number of constraints limited to a reasonable size. Figure 4.22 shows a number of cutting planes that surround the feasible region in a three-dimensional space. In the  $x_1$ - $x_2$  plane, the region ABCDEFA is found after three cutting planes are formed. When more cutting planes are formed, the feasible region narrows around the true minimum point. Although the cutting plane method can be used to tackle nonlinear objective functions by the above method, it is primarily used in problems with a linear objective function.

## 4.7 Feasible Direction Method

In the feasible direction method, linear approximations of the objective function and constraints are made to determine locally good search directions. The desired search direction at any point should be such that the points generated locally along that direction are feasible as well as better than the current point. Such search directions require two different considerations: feasibility and descent properties of NLP problems. The condition for a direction  $d$  to be descent is described in Chapter 3:  $\nabla f(x^{(t)}) \cdot d \leq 0$ . The condition for feasibility can be derived from similar considerations on active constraints. By including only the first-order term in Taylor's series expansion of an active constraint  $g_j(x)$  at any point  $x^{(t)}$  ( $\bar{j}$  represents all active constraints at the point  $x^{(t)}$ ), it can be shown that a direction  $d$  is feasible if the following condition is satisfied:

$$\nabla g_{\bar{j}}(x^{(t)}) \cdot d \geq 0. \quad (4.25)$$

Figure 4.23 shows a point  $x^{(t)}$  on a constraint surface  $g(x) \geq 0$ . The feasible region is shown hatched by dashed lines. Any direction  $d$  in the hatched region is a feasible direction.



**Figure 4.23** A feasible search direction.

The algorithm begins with a random point. At this point, the set of active constraints is found. If none of the constraints are active, the current point is an intermediate point in the search space and the steepest descent search direction is used. But if one or more search directions are active, the current point is on at least one constraint boundary. Thus, any arbitrary search direction (or the steepest descent direction) may not find a feasible point. Thus, a search direction which is maximally feasible and descent is found by solving an artificial LP problem. Once a search direction is found, a unidirectional search is performed along that direction to find the minimum point. This completes one iteration of Zoutendijk's feasible direction method (Zoutendijk, 1960).

### Algorithm

**Step 1** Set an iteration counter  $t = 0$ . Choose an initial feasible point  $x^{(0)}$  and a parameter for checking the constraint violation  $\epsilon$ .

**Step 2** At the current point  $x^{(t)}$ , let  $I^{(t)}$  be the set of indices of active constraints. In other words,

$$I^{(t)} = \{j : 0 \leq g_j(x^{(t)}) \leq \epsilon, j = 1, 2, \dots, J\}.$$

If  $I^{(t)}$  is empty, use  $d^{(t)} = \theta^{(t)} = -\nabla f(x^{(t)})$ , normalize  $d^{(t)}$  and go to Step 4.

**Step 3** Solve the following LP:

$$\text{Maximize } \theta$$

subject to

$$\begin{aligned} \nabla f(x^{(t)})d &\leq -\theta, \\ \nabla g_j(x^{(t)})d &\geq \theta, \quad j \in I^{(t)}; \\ -1 \leq d_i &\leq 1, \quad i = 1, 2, \dots, N. \end{aligned}$$

Label the solution  $d^{(t)}$  and  $\theta^{(t)}$ .

**Step 4** If  $\theta^{(t)} \leq 0$ , **Terminate**;

Else  $\bar{\alpha} = \min [\alpha : g_j(x^{(t)} + \alpha d^{(t)}) = 0, j = 1, 2, \dots, J, \text{ and } \alpha > 0]$ .

If no  $\bar{\alpha} > 0$  exists, set  $\bar{\alpha} = \infty$ .

**Step 5** Find  $\alpha^{(t)}$  such that

$$f(x^{(t)} + \alpha^{(t)}d^{(t)}) = \min[f(x^{(t)} + \alpha d^{(t)}) : 0 \leq \alpha \leq \bar{\alpha}].$$

Set  $x^{(t+1)} = x^{(t)} + \alpha^{(t)}d^{(t)}$ ,  $t = t + 1$ , and go to Step 2.

If none of the intermediate points is a boundary point, this method is similar to the steepest descent method (Cauchy's method), except that a modification is needed to take care of the constraint violations. However, in Zoutendijk's method, only a subset of constraints is used to define a subproblem. Thus, the LP problem is smaller than other linearization methods. One difficulty with this method is that since only active constraints are used to determine the search directions, the algorithm may result in zigzag iteration patterns, thereby making the convergence slower. We now present hand-calculations of a few iterations of this algorithm.

### EXERCISE 4.7.1

Consider the constrained Himmelblau function again:

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$\begin{aligned} g_1(x) &= 26 - (x_1 - 5)^2 - x_2^2 \geq 0, \\ g_2(x) &= 20 - 4x_1 - x_2 \geq 0, \\ x_1, x_2 &\geq 0. \end{aligned}$$

Let us recall that the minimum point of the above problem lies at  $x^* = (3, 2)^T$  with a function value equal to zero.

**Step 1** Let us choose an initial feasible point  $x^{(0)} = (0, 0)^T$  and a tolerance parameter  $\epsilon = 10^{-3}$ . We also set the iteration counter  $t = 0$ .

**Step 2** At this step, let us find the active constraints at point  $x^{(0)}$ . It turns out that only variable bounds are active. Calling these two inequality constraints  $g_3(x) = x_1 \geq 0$  and  $g_4(x) = x_2 \geq 0$ , we update the active constraint set  $I^{(0)} = \{3, 4\}$ . Since this set is not empty, we continue with Step 3.

**Step 3** At this step, we have to find a descent direction which is maximally away from both constraints  $g_3$  and  $g_4$ . At the initial point, both active constraints are orthogonal to each other. Thus, the desired search direction may make equal angle with all active constraints. But in any other situation, this may not be true. Thus, we find the optimal search direction by solving an LP problem. Calculating the derivative of the objective function and the active constraints at  $x^{(0)}$  numerically and denoting the search direction  $d = (d_1, d_2)^T$ , we obtain the following LP problem:

$$\text{Maximize } \theta$$

subject to

$$\begin{aligned} -14d_1 - 22d_2 &\leq -\theta, \\ d_1 &\geq \theta, \\ d_2 &\geq \theta, \\ -1 \leq d_1, d_2 &\leq 1. \end{aligned}$$

There exist a number of difficulties with the above formulation to be directly solved using the simplex method of LP technique. First of all, in the above formulation, the variables can take negative values, which are not allowed in a LP technique. Thus, we first substitute  $t_i = d_i + 1$  for  $i = 1, 2$  such that the variables  $t_i$  can take only positive values in the range  $(0, 2)$ . We rewrite the above LP problem in terms of the new variables:

$$\text{Maximize } \theta$$

subject to

$$14t_1 + 22t_2 - \theta \geq 36,$$

$$t_1 - \theta \geq 1,$$

$$t_2 - \theta \geq 1,$$

$$0 \leq t_1, t_2 \leq 2.$$

Secondly, the simplex method can handle only equality constraints. Slack variables are usually added or subtracted to convert inequality constraints to equality constraints. Therefore, for each of the above constraints we add a slack variable ( $y_1$  to  $y_5$ ). Thirdly, we observe that the problem variables and the slack variables do not constitute an initial basic feasible solution (refer to the Appendix for details). Thus, we introduce three more artificial variables ( $y_6$ ,  $y_7$ , and  $y_8$ ) to constitute an initial basic feasible solution for the first phase of the dual phase LP method (see Section A.3 for details). Thus, the underlying LP problem becomes as follows:

$$\left. \begin{array}{l} \text{Maximize } \theta \\ \text{subject to} \\ 14t_1 + 22t_2 - \theta - y_1 + y_6 = 36, \\ t_1 - \theta - y_2 + y_7 = 1, \\ t_2 - \theta - y_3 + y_8 = 1, \\ t_1 + y_4 = 2, \\ t_2 + y_5 = 2, \\ t_1, t_2, y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8 \geq 0. \end{array} \right\} \quad (4.26)$$

The three-variable problem of Exercise 4.6.1 now becomes an 11-variable problem. At first, we solve the above problem for the objective:

$$\text{Maximize } -(y_6 + y_7 + y_8). \quad (4.27)$$

Since all artificial variables must also be nonnegative, the solution to the above problem would have  $y_6 = y_7 = y_8 = 0$ , because the above objective function at this point would be zero. This solution will then be a candidate solution for the initial basic feasible solution of the problem presented in Equation (4.26). The dual phase LP method is described in the Appendix. The successive tables for the first problem of obtaining a feasible starting solution are shown in Tables 4.5 to 4.8.

**Table 4.5** The First Tableau for the First Phase of the Dual Phase LP Method

		0	0	0	0	0	0	0	-1	-1	-1	
$c_B$	Basic	$t_1$	$t_2$	$\theta$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$	$y_8$
-1	$y_6$	14	22	-1	-1	0	0	0	0	1	0	0
-1	$y_7$	1	0	-1	0	-1	0	0	0	0	1	0
-1	$y_8$	0	1	-1	0	0	-1	0	0	0	0	1
0	$y_4$	1	0	0	0	0	0	1	0	0	0	0
0	$y_5$	0	1	0	0	0	0	0	1	0	0	0
$(\Delta f)_q$		15	23	-3	-1	-1	-1	0	0	0	0	0
$f = -38$												

↑

The objective function value at Table 4.5 is  $f = -38$ . (Equation (4.27) is used as the objective.) In the table, it is clear that the nonbasic variable  $t_2$  corresponds to a maximum increase in the function value. (The quantity  $(\Delta f)_q$  is larger for  $t_2$ .) Thus, we choose  $t_2$  as the new basic variable.

It turns out from the minimum ratio rule (see the Appendix for the definition of the rule) that the basic variable  $y_8$  must be replaced by the variable  $t_2$  in the next iteration. We formulate the next row-echelon matrix. The calculation procedure for forming the row-echelon format is also discussed in the Appendix. The outcome of the calculation is shown in Table 4.6.

**Table 4.6** The Second Tableau for the First Phase of the Dual Phase LP Method

		0	0	0	0	0	0	0	-1	-1	-1	
$c_B$	Basic	$t_1$	$t_2$	$\theta$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$	$y_8$
-1	$y_6$	14	0	21	-1	0	22	0	0	1	0	-22
-1	$y_7$	1	0	-1	0	-1	0	0	0	0	1	0
0	$t_2$	0	1	-1	0	0	-1	0	0	0	0	1
0	$y_4$	1	0	0	0	0	0	1	0	0	0	0
0	$y_5$	0	0	1	0	0	1	0	1	0	0	-1
$(\Delta f)_q$		15	0	20	-1	-1	22	0	0	0	0	-23
$f = -15$												

↑

Note that the objective function value has improved considerably from the previous iteration. Here, we also observe that the nonbasic variable  $y_3$  corresponds to the maximum value of the quantity  $(\Delta f)_q$ . Thus, we choose  $y_3$  as the new basic variable. Using the minimum ratio rule, we also observe that the current basic variable  $y_6$  must be replaced by the variable  $y_3$  (Table 4.7). At the end of the third iteration, we observe that the nonbasic variable  $t_1$  must replace the basic variable  $y_7$ . The objective function value at this iteration

**Table 4.7** The Third Tableau for the First Phase of the Dual Phase LP Method

		0 0 0 0 0 0 0 -1 -1 -1										
$c_B$	Basic	$t_1$	$t_2$	$\theta$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$	$y_8$
0	$y_3$	0.64	0	0.95	-0.04	0	1	0	0	0.04	0	-1
-1	$y_7$		1.00	0	-1.00	0.00	-1	0	0	0.00	1	0
0	$t_2$	0.64	1	-0.04	-0.04	0	0	0	0	0.04	0	0
0	$y_4$	1.00	0	0.00	0.00	0	0	1	0	0.00	0	0
0	$y_5$	-0.64	0	0.04	0.04	0	0	0	1	-0.04	0	0
$(\Delta f)_q$		1.00	0	-1.00	0.00	-1	0	0	0	-1.00	0	-1
											$f = -1$	

↑

Ratios: 1 (first row), 1 (second row), 2.57 (third row), 2 (fourth row), and -ve (fifth row)

is  $f = -1$ . We form the next row-echelon matrix in Table 4.8. At this stage, we observe that all artificial variables are zero and the objective function is also equal to zero. This is the termination criterion for the first phase of the dual phase LP method. The solution of the above iteration is  $t_1 = 1$ ,  $t_2 = 1$ ,  $y_1 = 0$ ,  $y_2 = 0$ ,  $y_3 = 0$ ,  $y_4 = 1$ , and  $y_5 = 1$ . This solution was not obvious in the formulation of the problem presented in Equation (4.26).

**Table 4.8** The Fourth Tableau for the First Phase of the Dual Phase LP Method

		0 0 0 0 0 0 0 0 -1 -1 -1										
$c_B$	Basic	$t_1$	$t_2$	$\theta$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	$y_6$	$y_7$	$y_8$
0	$y_3$	0	0	1.59	-0.04	0.64	1	0	0	0.04	-0.64	-1
0	$t_1$	1	0	-1.00	0.00	-1.00	0	0	0	0.00	1.00	0
0	$t_2$	0	1	0.59	-0.04	0.64	0	0	0	0.04	-0.64	0
0	$y_4$	0	0	1.00	0.00	1.00	0	1	0	0.00	-1.00	0
0	$y_5$	0	0	-0.59	0.04	0.04	0	0	1	-0.04	0.64	0
$(\Delta f)_q$		0	0	0.00	0.00	0	0	0	0	-1.00	-1.00	-1
											$f = 0$	

We begin the second phase with the above solution as the initial solution. The objective in the second phase is to maximize the original function:  $f(x) = \theta$ . Since the artificial variables are no more required, we discontinue with them in subsequent computations. The row-echelon matrix is shown in Table 4.9. The objective function value at this iteration is  $f(x) = 0$ . Table 4.9 shows that the basic variable  $y_3$  must be replaced by the nonbasic variable  $\theta$ .

The new set of basic and nonbasic variables are formed in Table 4.10. At this iteration, the objective function value does not change, but the algorithm

**Table 4.9** The First Tableau for the Second Phase of the Dual Phase LP Method

		0	0	1	0	0	0	0	0
$c_B$	Basic	$t_1$	$t_2$	$\theta$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
0	$y_3$	0	0	1.59	-0.04	0.64	1	0	0
0	$t_1$	1	0	-1.00	0.00	-1.00	0	0	0
0	$t_2$	0	1	0.59	-0.04	0.64	0	0	0
0	$y_4$	0	0	1.00	0.00	1.00	0	1	0
0	$y_5$	0	0	-0.59	0.04	0.04	0	0	1
$(\Delta f)_q$		0	0	1.00	0.00	0.00	0	0	0
$f(x) = 0$									

↑

moves into a new solution. The table shows that the nonbasic variable  $y_1$  must become basic. The minimum ratio rule suggests that the basic variable  $y_4$  must be replaced with the new basic variable  $y_1$ .

**Table 4.10** The Second Tableau for the Second Phase of the Dual Phase LP Method

		0	0	1	0	0	0	0	0
$c_B$	Basic	$t_1$	$t_2$	$\theta$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$
1	$\theta$	0	0	1	-0.03	0.40	0.63	0	0
0	$t_1$	1	0	0	-0.03	-0.60	0.63	0	0
0	$t_2$	0	1	0	-0.03	0.40	-0.37	0	0
0	$y_4$	0	0	0	0.03	0.60	0.63	1	0
0	$y_5$	0	0	0	0.03	-0.40	0.37	0	1
$(\Delta f)_q$		0	0	0	0.03	-0.4	-0.63	0	0
$f(x) = 0$									

↑

It is important to note that even though the ratio calculated in the right-most column for the first row is zero, it is considered to be negative. In the implementation of the linear programming method, care should be taken to check the sign for both numerator and denominator. If they are of opposite sign, that row must be excluded from consideration. We form the new row-echelon matrix in Table 4.11. In this table, the quantity  $(\Delta f)_q$  corresponding to all nonbasic variables is nonpositive. Thus, we have obtained the optimum solution and we terminate the linear programming method. Thus, the final solution is  $t_1 = 2$ ,  $t_2 = 2$ ,  $\theta = 1$ ,  $y_1 = 35$ ,  $y_2 = 0$ ,  $y_3 = 0$ ,  $y_4 = 0$ , and  $y_5 = 0$ . These values satisfy all constraints in the problem presented in Equation (4.26). Now, we get back to Step 3 of the feasible direction search method.

**Table 4.11** The Third Tableau for the Second Phase of the Dual Phase LP Method

		0 0 1 0 0 0 0 0								
$c_B$	Basic	$t_1$	$t_2$	$\theta$	$y_1$	$y_2$	$y_3$	$y_4$	$y_5$	
1	$\theta$	0	0	1	0	1	0	1	0	1
0	$t_1$	0	0	0	0	0	0	1	0	2
0	$t_2$	0	1	0	0	1	-1	1	0	2
0	$y_1$	0	0	0	1	21	-22	1	0	35
0	$y_5$	0	0	0	0	-1	1	-1	1	0
$(\Delta f)_q$		0	0	0	0	-1	0	-1	0	$f(x) = 1$

**Step 3 (contd.)** The solution of the LP problem is  $t = (2, 2)^T$  or  $d^{(0)} = (1, 1)^T$  and  $\theta^{(0)} = 1$ . This solution implies that the resulting search direction makes equal angles with each of the two active constraints.

**Step 4** Since  $\theta^{(0)} = 1 > 0$ , we do not terminate the algorithm. Instead, we calculate the limits along the direction  $d^{(0)}$  before an infeasible point is found. Any generic point along  $d^{(0)}$  from  $x^{(0)}$  can be written as  $x(\alpha) = x^{(0)} + \alpha d^{(0)}$  or  $x(\alpha) = (\alpha, \alpha)^T$ . The upper limit on  $\alpha$  can be calculated by finding points along  $d^{(0)}$  that intersect with each constraint. The problem of finding the intersection of a straight line and any generic curve can be posed as a root-finding problem, which can be solved using an optimization algorithm discussed in Chapter 2. We substitute the expression for  $x_1 = \alpha$  and  $x_2 = \alpha$  in each constraint and then minimize the following problem:

$$\text{Minimize } \text{abs}[g_j(x(\alpha))]. \quad (4.28)$$

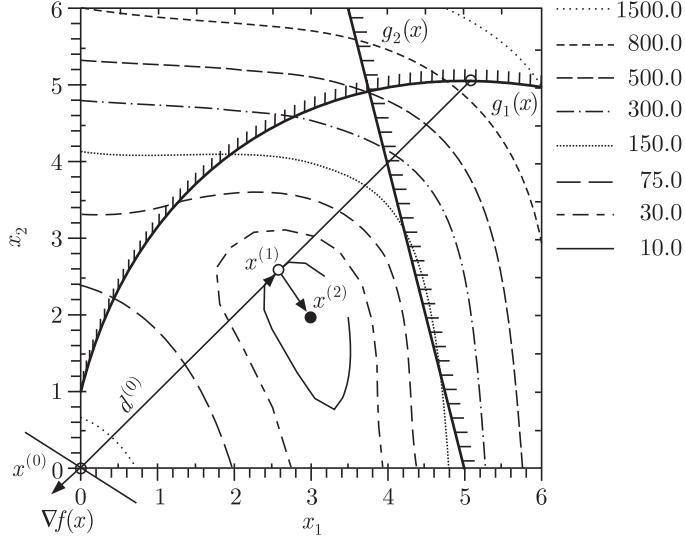
For example, the upper limit along  $d^{(0)}$  can be found for the first constraint by minimizing the unidirectional function:

$$\text{abs}[26 - (\alpha - 5)^2 - \alpha^2].$$

Since the absolute value of the argument is always considered, the above function allows only positive values. Since we are looking for points for which the constraint has a value zero, those points correspond to the minimum value of the above expression. Note that the problem described in Equation (4.28) is a single-variable function. Thus, we first bracket the minimum and then minimize the function. Using the bounding phase method from a starting point  $\alpha^{(0)} = 5$  and  $\Delta = 1$ , we obtain the bracketing interval  $(4, 6)$ . Next, we use the golden section search in that interval to obtain the minimum point with three decimal places of accuracy:  $\alpha_1^* = 5.098$ . The same solution can also be obtained by solving the quadratic expression  $g_1(x(\alpha)) = 0$ . Similarly, the limit on the second constraint can also be calculated:  $\alpha_2^* = 4.0$ . Other constraints

produce upper limits  $\alpha_3^* = \alpha_4^* = 0$ , which are not acceptable. Thus, the true upper limit is  $\bar{\alpha} = 4.0$ .

**Step 5** Once the lower and upper limit on  $\alpha$  are found, we perform another one dimensional search with the given objective function to find the minimum point along that direction. Using the golden section search, we obtain the minimum point in the interval  $(0, 4)$ :  $\alpha^* = 2.541$ , which corresponds to the new point  $x^{(1)} = (2.541, 2.541)^T$ . At this point, we increment the iteration counter and go to Step 2. This completes one iteration of the feasible direction method. The progress of this iteration is shown in Figure 4.24.



**Figure 4.24** A number of iterations of the feasible direction method.

**Step 2** At the new point, we find that no constraints are active. Thus,  $I^{(1)} = \emptyset$ , which means that the point is not on any constraint boundary and we are free to search in any direction locally. Therefore, we choose the steepest descent direction and the search direction is set according to the negative of the gradient of the objective function at the new point:

$$d^{(1)} = -\nabla f(x^{(1)}) = (16.323, -16.339)^T,$$

which is computed numerically. At this point the function value is  $f(x^{(1)}) = 8.0$ .

**Step 4** Once again, we compute the upper limit along the search direction  $d^{(1)}$ . Posing the root-finding problem as an optimization problem as shown in the previous iteration, we obtain the parameter  $\bar{\alpha} = \min[0.162, 0.149] = 0.149$ .

**Step 5** Performing a unidirectional search along  $d^{(1)}$  in the domain  $(0, 0.149)$ , we obtain  $\alpha^* = 0.029$ . The corresponding point is  $x^{(2)} = (3.018, 2.064)^T$  with an objective function value  $f(x^{(2)}) = 0.107$ .

This process continues until a point with a small derivative of the objective function is found. If the intermediate points fall on the constraint boundary frequently, this method may be expensive in terms of the overall computational time required to solve the problem. This situation may happen for problems with a narrow feasible region.

## 4.8 Quadratic Programming

In the previous sections, objective functions and constraints were linearized using different concepts and the resulting LP problems were solved in an iterative manner. In the quadratic programming (QP) approach, the objective function is approximated with a quadratic function; however, the constraints are still approximated linearly. With a quadratic objective function and linear constraints, the KKT optimality conditions will all become linear functions of the variables, as the equilibrium condition makes a derivative of the objective function and constraint functions. Since the KKT conditions of a QP problem are all linear, an LP solution methodology can be used to solve the KKT conditions. Moreover, since a quadratic problem with a positive semi-definite Hessian matrix is a convex problem, there will exist only one minimum point. Therefore, the solution of the KKT optimality condition is guaranteed to be the minimum solution to the QP problem.

For the nonlinear programming problem given in Equation (4.30), let us first make a variable transformation using

$$x'_i = x_i - x_i^{(L)}, \quad \text{for } i = 1, 2, \dots, n. \quad (4.29)$$

This is needed in order to ensure that all variable values take nonnegative values.

Thereafter, the objective function can be approximated to a quadratic function by using Taylor's series expansion principle and the constraints can be approximated to linear functions. The following quadratic problem can then be formed at the current point  $\mathbf{x}'^{(t)}$ :

$$\begin{aligned} \text{Minimize} \quad & q(\mathbf{x}'; \mathbf{x}'^{(t)}) = f(\mathbf{x}'^{(t)}) + \nabla f(\mathbf{x}'^{(t)})^T (\mathbf{x}' - \mathbf{x}'^{(t)}) \\ & + \frac{1}{2} (\mathbf{x}' - \mathbf{x}'^{(t)})^T H(\mathbf{x}'^{(t)}) (\mathbf{x}' - \mathbf{x}'^{(t)}), \\ \text{subject to} \quad & \tilde{g}_j(\mathbf{x}') \equiv g_j(\mathbf{x}'^{(t)}) + \nabla g_j(\mathbf{x}'^{(t)})^T (\mathbf{x}' - \mathbf{x}'^{(t)}) \geq 0, \quad j = 1, \dots, J; \\ & \tilde{h}_k(\mathbf{x}') \equiv h_k(\mathbf{x}'^{(t)}) + \nabla h_k(\mathbf{x}'^{(t)})^T (\mathbf{x}' - \mathbf{x}'^{(t)}) = 0, \quad k = 1, \dots, K. \\ & 0 \leq x'_i \leq (x_i^{(U)} - x_i^{(L)}), \quad i = 1, \dots, n. \end{aligned} \quad (4.30)$$

We shall now express the objective function and constraints in symbolic form. The upper variable bounds  $(x'_i \leq (x_i^{(U)} - x_i^{(L)}))$  are considered as

inequality constraints from here on. The linearized inequality constraints and the variable bounds can be written as  $A\mathbf{x}' \geq b$  with

$$A = \begin{bmatrix} \nabla g_1(\mathbf{x}'^{(t)})^T \\ \nabla g_2(\mathbf{x}'^{(t)})^T \\ \vdots \\ \nabla g_J(\mathbf{x}'^{(t)})^T \\ -I_1 \\ \vdots \\ -I_n \end{bmatrix}, \text{ and } b = \begin{bmatrix} -g_1(\mathbf{x}'^{(t)}) + \nabla g_1(\mathbf{x}'^{(t)})^T \mathbf{x}'^{(t)} \\ -g_2(\mathbf{x}'^{(t)}) + \nabla g_2(\mathbf{x}'^{(t)})^T \mathbf{x}'^{(t)} \\ \vdots \\ -g_J(\mathbf{x}'^{(t)}) + \nabla g_J(\mathbf{x}'^{(t)})^T \mathbf{x}'^{(t)} \\ x_1^{(U)} - x_1^{(L)} \\ \vdots \\ x_n^{(U)} - x_n^{(L)} \end{bmatrix},$$

where  $I_i$  is a zero vector of size  $n$  with  $i$ -th element equal to one. Note that the matrix  $A$  is of size  $(J+n) \times n$  and vector  $b$  is of size  $(J+n)$ . Also, we compute the  $C$ -matrix (of size  $K \times n$ ) and the  $d$ -vector (of size  $K$ ) as follows from the set of equality constraints:

$$C = \begin{bmatrix} \nabla h_1(\mathbf{x}'^{(t)})^T \\ \nabla h_2(\mathbf{x}'^{(t)})^T \\ \vdots \\ \nabla h_K(\mathbf{x}'^{(t)})^T \end{bmatrix}, \text{ and } d = \begin{bmatrix} -h_1(\mathbf{x}'^{(t)}) + \nabla h_1(\mathbf{x}'^{(t)})^T \mathbf{x}'^{(t)} \\ -h_2(\mathbf{x}'^{(t)}) + \nabla h_2(\mathbf{x}'^{(t)})^T \mathbf{x}'^{(t)} \\ \vdots \\ -h_K(\mathbf{x}'^{(t)}) + \nabla h_K(\mathbf{x}'^{(t)})^T \mathbf{x}'^{(t)} \end{bmatrix}.$$

Let us also consider the quadratic form of the objective function, as follows:

$$f(\mathbf{x}') = F + e^T \mathbf{x}'^{(t)} + \frac{1}{2} \mathbf{x}'^{(t)T} H \mathbf{x}', \quad (4.31)$$

where the constant term in the objective function is  $F = f(\mathbf{x}'^{(t)}) - \nabla f(\mathbf{x}'^{(t)})^T \mathbf{x}'^{(t)} + \frac{1}{2} \mathbf{x}'^{(t)T} H(\mathbf{x}'^{(t)}) \mathbf{x}'^{(t)}$ . The  $e$ -vector and  $H$ -matrix can be computed by comparing the above equation with the quadratic approximation of  $f(\mathbf{x}')$ . Ignoring the constant term, the  $e$ -vector can be written as follows:

$$e = \nabla f(\mathbf{x}'^{(t)}) - H(\mathbf{x}'^{(t)}) \mathbf{x}'^{(t)}. \quad (4.32)$$

The matrix  $H$  is simply the Hessian matrix of the quadratic approximation of  $f$  with respect to  $\mathbf{x}'$ . Thus, the quadratic programming problem becomes as follows:

$$\text{Minimize } F + e^T \mathbf{x}' + \frac{1}{2} \mathbf{x}'^{(t)T} H \mathbf{x}',$$

subject to

$$\left. \begin{array}{l} A\mathbf{x}' - b \geq 0, \\ C\mathbf{x}' - d = 0 \\ \mathbf{x}' \geq 0. \end{array} \right\} \quad (4.33)$$

The KKT optimality conditions for the above optimization problem can be written as follows:

$$e + H\mathbf{x}' - A^T\mu + C^T\omega - \nu = 0, \quad (4.34)$$

$$A\mathbf{x}' - b \geq 0, \quad (4.35)$$

$$C\mathbf{x}' - d = 0, \quad (4.36)$$

$$\mu_i(A\mathbf{x}' - b)_i = 0, \quad \nu_i(\mathbf{x}')_i = 0, \quad (4.37)$$

$$\mathbf{x}' \geq 0, \quad \mu \geq 0, \quad \nu \geq 0, \quad \omega \text{ free}. \quad (4.38)$$

Here, the Lagrange multipliers  $\mu \in R^{J+n}$ ,  $\omega \in R^K$ , and  $\nu \in R^n$ . The first Equation (4.35) is called the equilibrium condition, the second and third set of conditions (Equations (4.36) and (4.37)) are constraint satisfaction conditions, the fourth set of conditions (4.38) are called complementarity conditions and the fifth set of conditions (4.38) are strict nonnegativity of variables and Lagrange multipliers. Notice that there is no sign restriction for  $\omega$  parameters. A solution  $(\mathbf{x}', \mu, \omega, \nu)$  that satisfies all the above conditions is the solution to the quadratic programming problem given in Equation (4.33) under some regularity conditions. When the solution  $(\mathbf{x}'^*)$  to the above system of equations is found, the objective function  $f(\mathbf{x}'^*)$  can be computed using Equation (4.31).

Notice also that the above KKT conditions are all linear functions of variables  $(\mathbf{x}', \mu, \omega, \nu)$ . This implies that we may attempt to find a solution to the KKT conditions using a linear programming (LP) technique (Taha, 1989). However, an LP problem requires a linear objective function that is usually maximized and a set of linear equality or inequality constraints. Importantly, all variables of the LP must also take nonnegative values. The first three sets of conditions can be rewritten as follows:

$$\begin{aligned} H\mathbf{x}' - A^T\mu + C^T\omega - \nu + p &= -e, \\ A\mathbf{x}' - \lambda + q &= b, \\ C\mathbf{x}' + r &= d, \end{aligned} \quad (4.39)$$

where  $p \in R^n$ ,  $\lambda \in R^{J+n}$ , and  $r \in R^K$  are nonnegative slack variables and  $q \in R^{J+n}$  is a nonnegative artificial variable vector (see Section A.3). Note that for inequality constraints of the type  $A\mathbf{x}' - b \leq 0$ , the slack variables  $\lambda$  should be added (instead of subtracted, as in the above equation). Thus, the corresponding LP problem (for ' $\geq$ '-type inequality constraints) is given as follows:

$$\begin{aligned} \text{Maximize} \quad & -\sum_{i=1}^n p_i - \sum_{i=1}^{J+n} q_i - \sum_{i=1}^K r_i, \\ \text{subject to} \quad & H\mathbf{x}' - A^T\mu + C^T\omega - \nu + p = -e, \\ & A\mathbf{x}' - \lambda + q = b, \\ & C\mathbf{x}' + r = d, \\ & (\mathbf{x}', \mu, \nu, \lambda, p, q, r) \geq 0, \quad \omega \text{ free}, \end{aligned} \quad (4.40)$$

with the following conditions:

$$\mu_i \lambda_i = 0, \quad \text{for all } i = 1, 2, \dots, (J+n), \quad (4.41)$$

$$\nu_i x'_i = 0, \quad \text{for all } i = 1, 2, \dots, n. \quad (4.42)$$

As mentioned, for ' $\leq$ '-type inequality constraints ( $A\mathbf{x}' \leq b$ ), the second equation becomes

$$A\mathbf{x}' + \lambda + q = b,$$

and also the sign of  $A^T \mu$  term must be positive in the first set of constraints in the above optimization problem. Both  $\lambda$  and  $q$  variables are needed to be added in the left-hand side part of the equation. However, the cost coefficient for  $q$  variables is only considered nonzero.

The usual simplex method (discussed in Section A.2 in the Appendix) does not automatically satisfy conditions given in Equations (4.41) and (4.42) among the variables. Thus, the simplex method needs to be changed somewhat to satisfy these conditions. While we use the simplex method, the following two sets of conditions must be satisfied when entering a new variable in the set of basic variables. Since the basic variables take non-zero values, we need to restrict two complementary variables to exist in the basic variable set. In other words, if  $\lambda_k$  is present in the basic variable set, even if  $\mu_k$  variable is decided to enter the basic set from maximum  $(\Delta f)_q$  consideration, we shall not accept this variable. Instead, the non-basic variable having the next highest  $(\Delta f)_q$  will be chosen, provided its complementary variable does not exist in the basic variable set. This process will continue till all the above conditions are satisfied within the basic variables. We call this method as the *conditioned simplex* method.

After the solution  $\mathbf{x}'^*$  is found, the objective function must be computed using Equation (4.31).

### EXERCISE 4.8.1

We illustrate the working of the quadratic programming procedure to an example problem, given below:

$$\begin{aligned} \text{Minimize} \quad & f(\mathbf{x}) = 2x_1^2 + x_2^2 - 4x_1 - 2x_2 - 2x_1x_2, \\ \text{subject to} \quad & g_1(\mathbf{x}) = x_1 + x_2 \leq 2, \\ & g_2(\mathbf{x}) = 2x_2 - x_1 = 0, \\ & x_1, x_2 \geq 0. \end{aligned}$$

This is a quadratic problem as the objective function is quadratic and both inequality and equality constraints are linear. Let us now write the vectors

and matrices relevant to the above problem:

$$H = \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix}, \quad e = \begin{bmatrix} -4 \\ -2 \end{bmatrix},$$

$$A = [1 \ 2], \quad C = [-1 \ 2].$$

Also,  $b = 2$  and  $d = 0$ . Since the lower bound on both  $x_1$  and  $x_2$  is zero,  $\mathbf{x}' = \mathbf{x}$ . We also treat the inequality constraint as ' $\leq$ '-type. We write the LP constraints, as follows:

$$H\mathbf{x} + A^T\mu + C^T\omega - \nu + p = -e,$$

$$A\mathbf{x} + \lambda + q = b,$$

$$C\mathbf{x} + r = d,$$

$$(\mathbf{x}, \mu, \omega, \nu, \lambda, p, q, r) \geq 0,$$

Here,  $p = (p_1, p_2)^T$ ,  $q \in R^1$ ,  $r \in R^1$  and  $\lambda \in R^1$  are slack variables and  $\mu \in R^1$ ,  $\omega \in R^1$  and  $\nu \in R^2$  are Lagrange multipliers. We now formulate the LP problem:

$$\text{Maximize} \quad -p_1 - p_2 - q - r$$

subject to

$$\begin{aligned} \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \mu + \begin{bmatrix} -1 \\ 2 \end{bmatrix} \omega - \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} + \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} &= -\begin{bmatrix} -4 \\ -2 \end{bmatrix}, \\ [1 \ 1] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \lambda + q &= 2, \\ [-1 \ 2] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + r &= 0, \\ (x_1, x_2, \mu, \nu_1, \nu_2, \lambda, p_1, p_2, q, r) &\geq 0, \quad \omega \text{ free}, \end{aligned} \tag{4.43}$$

with the following complementarity conditions:

$$\mu\lambda = 0, \quad \nu_1 x_1 = 0, \quad \nu_2 x_2 = 0. \tag{4.44}$$

The initial simplex is presented in Table 4.12.

**Table 4.12** Initial Simplex for Exercise Problem 4.8.1

		0 0 0 0 0 0 -1 -1 -1 -1										
$c_B$	Basic	$x_1$	$x_2$	$\mu$	$\omega$	$\nu_1$	$\nu_2$	$\lambda$	$p_1$	$p_2$	$q$	$r$
-1	$p_1$	4	-2	1	-1	-1	0	0	1	0	0	0
-1	$p_2$	-2	2	1	2	0	-1	0	0	1	0	0
-1	$q$	1	1	0	0	0	0	1	0	0	1	0
-1	$r$	-1	2	0	0	0	0	0	0	0	1	0
$(\Delta f)_q$		2	3	2	1	-1	-1	1	0	0	0	0
											$f(x) = -8$	

↑

The basic variables are  $p_1, p_2, q$  and  $r$ . It can be observed that the variable  $r$  will get replaced by  $x_2$  in the next tableau (Table 4.13).

**Table 4.13** Second Simplex for the Exercise Problem 4.8.1

		0 0 0 0 0 0 -1 -1 -1 -1										
$c_B$	Basic	$x_1$	$x_2$	$\mu$	$\omega$	$\nu_1$	$\nu_2$	$\lambda$	$p_1$	$p_2$	$q$	$r$
-1	$p_1$	3	0	1	-1	-1	0	0	1	0	0	1
-1	$p_2$	-1	0	1	2	0	-1	0	0	1	0	-1
-1	$q$	1.5	0	0	0	0	0	1	0	0	1	-0.5
0	$x_2$	-0.5	1	0	0	0	0	0	0	0	0	0.5
$(\Delta f)_q$		3.5	0	2	1	-1	-1	1	0	0	0	-1.5
											$f(x) = -8$	

↑

Notice that complementarity conditions are satisfied. The solution here is  $(x_1, x_2)^T = (0, 0)^T$ . The variable  $p_1$  gets replaced by  $x_1$ . Table 4.14 shows the next tableau.

**Table 4.14** Third Simplex for the Exercise Problem 4.8.1

		0 0 0 0 0 0 -1 -1 -1 -1										
$c_B$	Basic	$x_1$	$x_2$	$\mu$	$\omega$	$\nu_1$	$\nu_2$	$\lambda$	$p_1$	$p_2$	$q$	$r$
0	$x_1$	1	0	0.33	-0.33	-0.33	0	0	0.33	0	0	0.33
-1	$p_2$	0	0	1.33	1.67	-0.33	-1	0	0.33	2	0	-0.67
-1	$q$	0	0	-0.5	0.5	0.5	0	1	-0.5	0	1	-1
0	$x_2$	0	1	0.17	-0.17	-0.17	0	0	0.17	0	0	0.67
$(\Delta f)_q$		0	0	0.83	2.17	0.17	-1	1	-1.17	0	0	-2.67
											$f(x) = -3.33$	

↑

The solution here is  $(x_1, x_2)^T = (1.33, 0.67)^T$ . Now,  $q$  gets replaced by  $\omega$ . The new tableau is shown in Table 4.15.

**Table 4.15** Fourth Simplex for the Exercise Problem 4.81

		0 0 0 0 0 0 -1 -1 -1 -1										
$c_B$	Basic	$x_1$	$x_2$	$\mu$	$\omega$	$\nu_1$	$\nu_2$	$\lambda$	$p_1$	$p_2$	$q$	$r$
0	$x_1$	1	0	0	0	0	0	0.67	0	0	0.67	-0.33
-1	$p_2$	0	0	3	0	-2	-1	-3.33	2	1	-3.33	2.67
0	$\omega$	0	0	-1	1	1	0	2	-1	0	2	-2
0	$x_2$	0	1	0	0	0	0	0.33	0	0	0.33	0.33
$(\Delta f)_q$		0	0	3	0	-2	-1	-3.33	1	0	-4.33	1.67
										$f(x) = -3.33$		

↑

The solution here is still  $(x_1, x_2)^T = (1.33, 0.67)^T$ . Now,  $p_2$  gets replaced by  $\mu$ . Table 4.16 shows the tableau.

**Table 4.16** Fifth Simplex for the Exercise Problem 4.8.1

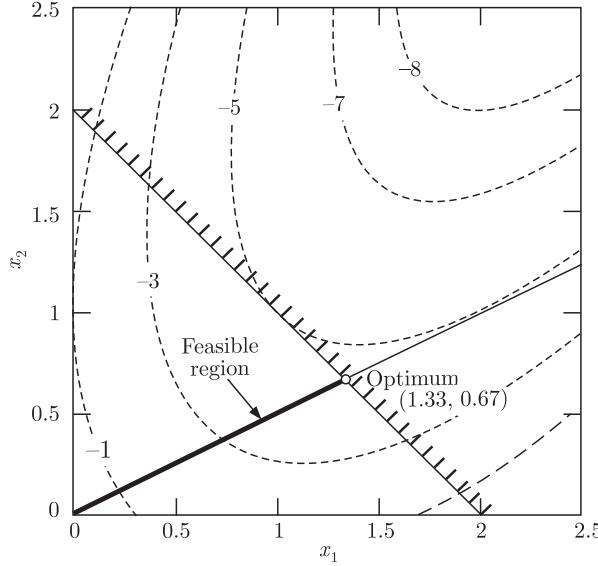
		0 0 0 0 0 0 -1 -1 -1 -1										
$c_B$	Basic	$x_1$	$x_2$	$\mu$	$\omega$	$\nu_1$	$\nu_2$	$\lambda$	$p_1$	$p_2$	$q$	$r$
0	$x_1$	1	0	0	0	0	0	0.67	0	0	0.67	-0.33
0	$\mu$	0	0	1	0	-0.67	-0.33	-1.11	0.67	0.33	-1.11	0.89
0	$\omega$	0	0	0	1	0.33	-0.33	0.89	-0.33	0.33	0.89	-1.11
0	$x_2$	0	1	0	0	0	0	0.33	0	0	0.33	0.33
$(\Delta f)_q$		0	0	0	0	0	0	-1	-1	-1	-1	
										$f(x) = 0$		

Finally, the solution here is  $(x_1, x_2)^T = (1.33, 0.67)^T$ . The objective function value computed at  $(x_1, x_2)^T$  using the original objective function  $f(\mathbf{x})$  is -4.44. Notice that in all the above simplexes, complementarity conditions (Equation 4.44) are satisfied, that is, in no tableau, both elements of the pairs ( $\mu$  and  $q$ ) or ( $\nu_1$  and  $x_1$ ) or ( $\nu_2$  and  $x_2$ ) are present in the basic variable set.

Figure 4.25 shows the feasible region and the optimal solution graphically. It is clear that an identical solution is obtained by the QP method in one iteration. Interestingly, while solving a QP problem using the above QP method, no initial starting solution needs to be supplied. The resulting LP problem chooses a basic feasible solution and the algorithm is started.

#### 4.8.1 Sequential Quadratic Programming

For solving a quadratic programming problem having a quadratic objective function and linear constraints, the approximations mentioned above will result in original functions, as shown in the above example problem. Hence, the solution of the conditioned LP will result in the optimum solution of the QP problem. However, if the original problem is not a QP, the above procedure can be applied in sequence. This method is known as the *sequential quadratic programming* (SQP) method, which we describe next.



**Figure 4.25** Quadratic programming problem is illustrated.

### Algorithm

**Step 1** Set an iteration counter  $t = 0$ . Choose an initial feasible point  $x^{(0)}$  and two termination parameters  $\epsilon_1$  and  $\epsilon_2$ .

**Step 2** At the current point  $x^{(t)}$ , make a coordinate transformation (in terms of  $\mathbf{x}'$ ) as shown in Equation (4.29) and then a quadratic approximation of  $f(\mathbf{x}')$ . Also, make linear approximations of  $g_j(\mathbf{x}')$  and  $h_k(\mathbf{x}')$  at the point  $\mathbf{x}'^{(t)}$ .

**Step 3** Formulate the LP given in Equation (4.33) and solve using the conditioned simplex method described above. Label the solution  $\mathbf{y}^{(t)}$ .

**Step 4** If  $\|\mathbf{y}^{(t)} - \mathbf{x}'^{(t)}\| \leq \epsilon_1$  and  $|f(\mathbf{y}^{(t)}) - f(\mathbf{x}'^{(t)})| \leq \epsilon_2$ , **Terminate**; Else increment counter  $t = t + 1$  and go to Step 2.

### EXERCISE 4.8.2

Consider the constrained Himmelblau function again:

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2 - 7)^2$$

subject to

$$g_1(x) = (x_1 - 5)^2 + x_2^2 \leq 26,$$

$$g_2(x) = 4x_1 + x_2 \leq 20,$$

$$x_1, x_2 \geq 0.$$

Let us recall that the minimum point of the above problem lies at  $x^* = (3, 2)^T$  with a function value equal to zero.

**Step 1** We choose an initial point  $x^{(0)} = (1, 1)^T$  and set iteration counter  $t = 0$ . Also, we set  $\epsilon_1 = \epsilon_2 = 0.001$  for termination.

**Step 2** The above problem (with  $n = 2$ ,  $J = 2$ , and  $K = 0$ ) does not have a specified upper bound on both variables. Thus, parameters  $\mu$  and  $\lambda$ -vectors have two ( $J = 2$ ) elements each. Since the lower-bound of both variables is also zero, no variable transformation is needed for this problem. Thus,  $\mathbf{x}'^{(t)} = \mathbf{xt}$  at every  $t$ .

The quadratic approximation of  $f(\mathbf{x})$  at  $x^{(0)}$  yields the following  $e$ -vector and  $H$ -matrix:

$$e = \begin{bmatrix} -28 \\ -36 \end{bmatrix}, \quad H = \begin{bmatrix} -26 & 8 \\ 8 & -10 \end{bmatrix}.$$

Linear approximation of both inequality constraints yield the following  $A$ -matrix and  $b$ -vector:

$$A = \begin{bmatrix} -8 & 2 \\ 4 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 20 \end{bmatrix}.$$

The following LP problem is then formulated:

$$\text{Minimize} \quad -p_1 - p_2 - q_1 - q_2,$$

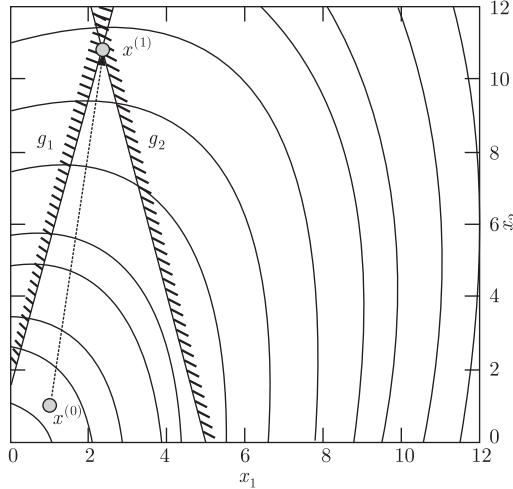
subject to

$$\left. \begin{array}{l} H\mathbf{x} + A^T\mu - \nu + p = -e, \\ A\mathbf{x} + \lambda + q = b, \\ (\mathbf{x}, \mu, \nu, \lambda, p, q) \geq 0, \end{array} \right\} \quad (4.45)$$

with following conditions:

$$\begin{aligned} \mu_1\lambda_1 &= 0, & \mu_2\lambda_2 &= 0, \\ \nu_1x_1 &= 0, & \nu_2x_2 &= 0. \end{aligned}$$

Figure 4.26 shows the two linear inequality constraints and the contour of the quadratic objective function at this iteration. The optimal solution  $((2.312, 10.750)^T)$  to this quadratic programming problem is shown in the figure. However, we show the tableau of the conditioned simplex method to solve the above problem in Table 4.17. First, variables  $p_1$ ,  $p_2$ ,  $q_1$  and  $q_2$  are basic variables.



**Figure 4.26** LP problem for the first iteration of the QP method is shown.

**Table 4.17** Initial Simplex for the Exercise Problem 4.8.2

		0	0	0	0	0	0	0	-1	-1	-1	-1	
$c_B$	Basic	$x_1$	$x_2$	$\mu_1$	$\mu_2$	$\nu_1$	$\nu_2$	$\lambda_1$	$\lambda_2$	$p_1$	$p_2$	$q_1$	$q_2$
-1	$p_1$	-26	8 -8	4	-1	0	0	0	1	0	0	0	28
-1	$p_2$		8 -10	2	1	0	-1	0	0	0	1	0	36
-1	$q_1$	-8	2	0	0	0	0	1	0	0	0	1	0
-1	$q_2$	4	1	0	0	0	0	0	1	0	0	0	20
$(\Delta f)_q$		-22	1 -6	5	-1	-1	1	1	0	0	0	0	$f(x) = -87$

↑

After this iteration,  $\mu_2$  enters the basic variable set and  $p_2$  becomes non-basic. Since  $\lambda_2$  is not present in the basic variable set, this inclusion is allowed. Iteration 2 is applied next and the simplex is shown in Table 4.18.

**Table 4.18** Initial Simplex for the Exercise Problem 4.8.2

		0	0	0	0	0	0	0	0	-1	-1	-1	-1
$c_B$	Basic	$x_1$	$x_2$	$\mu_1$	$\mu_2$	$\nu_1$	$\nu_2$	$\lambda_1$	$\lambda_2$	$p_1$	$p_2$	$q_1$	$q_2$
-1	$\mu_2$	-6.5	2 -2	1	-0.25	0	0	0	0.25	0	0	0	7
-1	$p_2$	14.5	-12	4	0	0.25	-1	0	0	-0.25	1	0	29
-1	$q_1$		-8	2	0	0	0	0	1	0	0	1	0
-1	$q_2$	4	1	0	0	0	0	0	1	0	0	0	20
$(\Delta f)_q$		10.5	-9	4	0	0.25	-1	1	1	-1.25	0	0	0

↑

Now,  $x_1$  enters the basic variable set. Since  $\nu_1$  does not exist in the basic variable set, this is also allowed. Notice how the function value increases from  $-87$  to  $-52$  in one iteration of the simplex method. Iteration 3 is tabulated in Table 4.19.

**Table 4.19** Third Simplex for the Exercise Problem 4.8.2

0 0 0 0 0 0 0 0 -1 -1 -1 -1													
$c_B$	Basic	$x_1$	$x_2$	$\mu_1$	$\mu_2$	$\nu_1$	$\nu_2$	$\lambda_1$	$\lambda_2$	$p_1$	$p_2$	$q_1$	$q_2$
0	$\mu_2$	0	-3.38	-0.21	1	-0.14	-0.45	0	0	0.14	0.48	0	0
0	$x_1$	1	-0.83	0.28	0	0.02	-0.07	0	0	-0.02	0.07	0	0
-1	$q_1$	0	-4.62	2.21	0	0.14	-0.55	1	0	-0.14	0.55	1	0
-1	$q_2$	0	4.31	-1.10	0	-0.07	0.28	0	1	0.07	-0.28	0	1
$(\Delta f)_q$		0	-0.31	1.10	0	0.07	-0.28	1	1	-1.07	-0.72	0	0
$f(x) = -31$													

↑

Parameter  $\mu_1$  replaces  $x_1$  in this iteration. Iteration 4 is executed and is shown in Table 4.20.

**Table 4.20** Fourth Simplex for the Exercise Problem 4.8.2

0 0 0 0 0 0 0 0 -1 -1 -1 -1													
$c_B$	Basic	$x_1$	$x_2$	$\mu_1$	$\mu_2$	$\nu_1$	$\nu_2$	$\lambda_1$	$\lambda_2$	$p_1$	$p_2$	$q_1$	$q_2$
0	$\mu_2$	0.75	-4	0	1	-0.13	-0.5	0	0	0.13	0.5	0	0
0	$\mu_1$	3.63	-3	1	0	0.62	-0.25	0	0	-0.06	0.25	0	0
-1	$q_1$	-8	2	0	0	0	0	1	0	0	0	1	0
-1	$q_2$	4	1	0	0	0	0	0	1	0	0	0	1
$(\Delta f)_q$		-4	3	0	0	0	0	0	1	1	-1	-1	0
$f(x) = -23$													

↑

Now,  $x_2$  in place of  $q_1$  in the basic variable set. Again, this move is allowed. The function value increases to  $-23$ . Iteration 5 is next and is shown in Table 4.21.

**Table 4.21** Fifth Simplex for the Exercise Problem 4.8.2

0 0 0 0 0 0 0 0 -1 -1 -1 -1													
$c_B$	Basic	$x_1$	$x_2$	$\mu_1$	$\mu_2$	$\nu_1$	$\nu_2$	$\lambda_1$	$\lambda_2$	$p_1$	$p_2$	$q_1$	$q_2$
0	$\mu_2$	-15.25	0	0	1	-0.13	-0.5	2	0	0.13	0.5	2	0
0	$\mu_1$	-8.38	0	1	0	0.06	-0.25	1.5	0	-0.06	0.25	1.5	0
0	$x_2$	-4	1	0	0	0	0	0.5	0	0	0	0.5	0
-1	$q_2$	8	0	0	0	0	0	-0.5	1	0	0	-0.5	1
$(\Delta f)_q$		8	0	0	0	0	0	-0.5	1	-1	-1	-1.5	0
$f(x) = -18.5$													

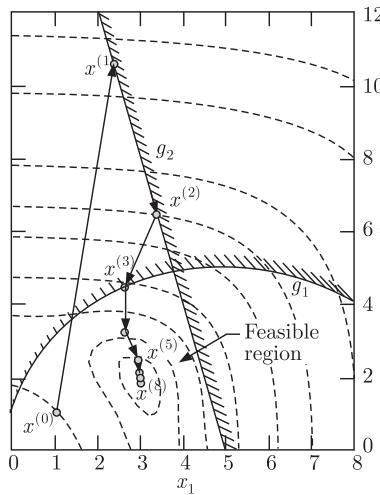
↑

Next, variable  $x_1$  enters the basic variable set. Iteration 6 is then executed and is shown in Table 4.22.

**Table 4.22** Sixth Simplex for the Exercise Problem 4.8.2

$c_B$	Basic	$x_1$	$x_2$	$\mu_1$	$\mu_2$	$\nu_1$	$\nu_2$	$\lambda_1$	$\lambda_2$	$p_1$	$p_2$	$q_1$	$q_2$	
0	$\mu_2$	0	0	0	1	-0.13	-0.5	1.05	1.91	0.13	0.5	1.05	1.91	62.77
0	$\mu_1$	0	0	1	0	0.06	-0.25	0.98	1.05	-0.06	0.25	0.98	1.05	31.12
0	$x_2$	0	1	0	0	0	0	0.25	0.5	0	0	0.25	0.5	10.75
0	$x_1$	1	0	0	0	0	0	-0.06	0.13	0	0	0	-0.06	0.13
$(\Delta f)_q$														$f(x) = 0$

Since there is no positive value of  $(\Delta f)_q$ , the simplex algorithm terminates. Also, the simplex function value is zero, which is the optimum value. The solution is  $x^{(1)} = (2.31, 10.75)^T$ , which is identical to that found in Figure 4.26. At this solution, other parameters are as follows:  $\nu = (0, 0)^T$ ,  $\mu = (62.77, 31.12)^T$ , and  $\lambda = (0, 0)^T$ . Notice at this solution the complementarity conditions are satisfied. The solution  $\mathbf{x}^{(1)} = (2.31, 10.75)^T$  is shown in Figure 4.26 and also in Figure 4.27.



**Figure 4.27** A sequential application of QP algorithm finds the constrained minimum of the example problem.

**Step 3** Since there is a substantial difference between solutions  $\mathbf{x}^{(0)}$  and  $\mathbf{x}^{(1)}$ , we do not terminate the algorithm and proceed to Step 2 of the algorithm with  $\mathbf{x}^{(1)}$ .

**Step 2** The LP is formed at  $\mathbf{x}^{(1)}$  and is solved. The solution is found to be  $\mathbf{x}^{(2)} = (3.372, 6.513)^T$ . This solution is marked on Figure 4.27.

**Step 3** Due to substantial difference in  $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$ , we do not terminate the procedure and move to Step 2.

This procedure continues and eventually converges to the constrained minimum solution  $\mathbf{x}^* = (3, 2)^T$  after eight iterations, as shown in Table 4.23. The progress of the algorithm is shown in Figure 4.27.

**Table 4.23** Progress of Quadratic Programming Algorithm on the Example Problem

$t$	$x_1^{(t)}$	$x_2^{(t)}$	$f(x^{(t)})$	$g_1(x^{(t)})$	$g_2(x^{(t)})$
0	1.000	1.000	106.000	-9.000	-15.000
1	2.312	10.750	12319.257	96.785	0.000
2	3.372	6.513	1552.257	19.073	0.000
3	2.615	4.510	254.744	0.033	-5.031
4	2.623	3.247	38.762	-9.807	-6.262
5	2.899	2.483	4.270	-15.421	-5.922
6	2.973	2.116	0.205	-17.414	-5.993
7	2.998	2.009	0.001	-17.955	-5.999
8	3.000	2.000	0.000	-18.000	-6.000

Most optimization problems can be assumed to have a quadratic objective function in the neighbourhood of the optimal solution. Thus, when a near-optimal solution is obtained, the use of SQP procedure may prove to be beneficial. SQP method is popularly coded in many commercial optimization softwares.

## 4.9 Generalized Reduced Gradient Method

The reduced gradient method is an implicit variable elimination algorithm for solving NLP problems. We first discuss the reduced gradient method for handling equality constraints and later present a generalized reduced gradient method for handling inequality constraints.

For an NLP problem having  $N$  design variables and  $K$  (where  $K < N$ ) equality constraints (linear or nonlinear) and no inequality constraints, any  $K$  decision variables can be eliminated using the equality constraints. Thus, an unconstrained optimization problem can be formulated with only  $(N - K)$  decision variables. One such variable elimination algorithm has been described in Section 4.5.1. But the method discussed here is more elegant and can also be applied to wider classes of problems including problems having inequality constraints. However, the basic working principle is the same. A unconstrained function is first formed with a reduced number of  $(N - K)$  variables. Then, the unconstrained function is optimized to find a solution for the chosen  $(N - K)$  design variables. Thereafter, optimal values for other  $K$  variables which are dependent on the chosen  $(N - K)$  variables are obtained using the equality constraints.

The  $K$  dependent variables which are eliminated from the optimization process are known as *basic* variables ( $\hat{x}$ ) and the remaining  $(N - K)$  variables are known as *nonbasic* variables ( $\bar{x}$ ). Using Taylor's series expansion of the

objective function and substituting the basic variables in terms of nonbasic variables, the unconstrained objective function can be written in terms of two matrices  $J$  and  $C$  at a point  $x^{(t)}$  as follows:

$$\nabla \tilde{f}(x^{(t)}) = \nabla \bar{f}(x^{(t)}) - \nabla \hat{f}(x^{(t)}) J^{-1} C, \quad (4.46)$$

where

$$J = \begin{pmatrix} \nabla \hat{h}_1 \\ \nabla \hat{h}_2 \\ \vdots \\ \nabla \hat{h}_K \end{pmatrix}, \quad C = \begin{pmatrix} \nabla \bar{h}_1 \\ \nabla \bar{h}_2 \\ \vdots \\ \nabla \bar{h}_K \end{pmatrix}.$$

The matrix  $J$  is a  $K \times K$  matrix and the matrix  $C$  is a  $K \times (N - K)$  matrix evaluated at  $x^{(t)}$ . The vector  $\nabla \hat{h}_k$  is calculated by differentiating the  $k$ -th constraint with respect to basic variables  $\hat{x}$ :

$$\nabla \hat{h}_k = \left( \frac{\partial h_k(x)}{\partial \hat{x}_1}, \frac{\partial h_k(x)}{\partial \hat{x}_2}, \dots, \frac{\partial h_k(x)}{\partial \hat{x}_K} \right).$$

Thus, each vector  $\nabla \hat{h}_k$  is a  $1 \times K$  vector. Similarly the vector  $\nabla \bar{h}_k$  is calculated by differentiating  $k$ -th constraint with respect to nonbasic variables  $\bar{x}$ :

$$\nabla \bar{h}_k = \left( \frac{\partial h_k(x)}{\partial \bar{x}_1}, \frac{\partial h_k(x)}{\partial \bar{x}_2}, \dots, \frac{\partial h_k(x)}{\partial \bar{x}_{N-K}} \right).$$

Thus, the above vector is a  $1 \times (N - K)$  vector. The notation used in the above expressions may look intimidating, but they are simply vectors of real numbers computed using the numerical differentiation technique given in Equation (3.4). The vector  $\nabla \bar{f}(x^{(t)})$  is a part of the gradient vector of the objective function that corresponds to the nonbasic variables evaluated at the point  $x^{(t)}$ . Similarly, the vector  $\nabla \hat{f}(x^{(t)})$  is rest of the gradient vector of the objective function. The computation of these matrices and vectors will be clear when we simulate this algorithm on an exercise problem. The term  $\nabla f(x^{(t)})$  is known as the *reduced gradient* at the point  $x^{(t)}$ . It can be shown that the reduced gradient will be zero at the constrained optimum point.

In the generalized reduced gradient algorithm, inequality constraints are handled by introducing a slack variable for each constraint. An inequality constraint  $g_j(x) \geq 0$  can be transformed into an equality constraint by introducing a variable  $x_{N+j}$ :

$$g_j(x) - x_{N+j} = 0.$$

The variable  $x_{N+j}$  takes positive values for feasible points and takes negative values for infeasible points. If for a point  $x_{N+j}$  is zero, the point lies on the boundary. Thus, by adding an extra variable, each inequality constraint can

be transformed into an equality constraint and the original reduced gradient algorithm can be used. This procedure of handling inequality constraints is known as the *slack variable* strategy. Later, we shall discuss an artificial variable strategy, which can also be used to handle inequality constraints.

Although the basic structure of the algorithm is simple and straightforward, there remain some fundamental questions to be answered. First, in the beginning of the algorithm one has to choose  $(N - K)$  nonbasic variables from  $N$  decision variables. Since nonbasic variables are used in the optimization process (and basic variables are derived from the nonbasic variables), they are to be so chosen that they do not lie on the constraint or variable boundaries. Secondly, the new point obtained using the search algorithm may not satisfy all  $K$  equality constraints. This is because the solution involves only  $(N - K)$  variables. The remaining  $K$  variables are derived from solving linear approximations of  $K$  equality constraints. If all constraints are linear, there is no difficulty. In case of nonlinear constraints, the solution of the basic variables is not guaranteed to satisfy equality constraints, thereby making the solution infeasible. Thus,  $K$  basic variables can be manipulated to find a feasible point that would satisfy all equality constraints.

Therefore, every time the optimization algorithm creates a new set of nonbasic variables, the corresponding basic variables are found by solving linearized constraints. Thereafter, the feasibility of the solution is checked. If the point is not found to be feasible, it is made so by changing the basic variables in a systematic way.

Let us say that at a point  $v^{(t)}$ , the basic variables are represented by  $\hat{v}^{(t)}$ . A feasible solution must lie on the surface of intersection of all constraints. This solution  $v^{(t)} \equiv (\hat{v}^{(t)}, \bar{v}^{(t)})$  may not satisfy all  $K$  equality constraints. Thus, we have to find a modified point which will satisfy all constraints. This is a *zero-finding* problem which can be solved using the Newton-Raphson method (or any other unconstrained optimization method), as discussed in Chapter 2. In order to find a point satisfying  $h_k(v) = 0$ ,  $k = 1, 2, \dots, K$ , we can use the following recursion equation from the initial point  $v^{(t)}$ :

$$v^{(t+1)} = v^{(t)} - \nabla h^{-1}(v^{(t)}) \cdot h(v^{(t)}), \quad (4.47)$$

where the second term in the right side expression is

$$\left[ \begin{array}{cccc} \frac{\partial h_1}{\partial \hat{x}_1} & \frac{\partial h_2}{\partial \hat{x}_1} & \dots & \frac{\partial h_K}{\partial \hat{x}_1} \\ \frac{\partial h_1}{\partial \hat{x}_2} & \frac{\partial h_2}{\partial \hat{x}_2} & \dots & \frac{\partial h_K}{\partial \hat{x}_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_1}{\partial \hat{x}_K} & \frac{\partial h_2}{\partial \hat{x}_K} & \dots & \frac{\partial h_K}{\partial \hat{x}_K} \end{array} \right]_{v^{(t)}}^{-1} \cdot \begin{Bmatrix} h_1(v^{(t)}) \\ h_2(v^{(t)}) \\ \vdots \\ h_K(v^{(t)}) \end{Bmatrix}.$$

Since this process involves calculation of inverse of a  $K \times K$  matrix and since a number of iterations of the above equation are usually required to get a feasible point, this method is computationally expensive. In the following, we outline the reduced gradient algorithm and then present the generalized version to handle nonlinear constraints.

### Algorithm

**Step 1** Choose an initial point  $x^{(0)}$ , a termination factor  $\epsilon$ . Set  $t = 0$ . Transform all inequality constraints into equality constraints by adding slack variables.

**Step 2** Calculate the following quantity for all variables,  $i = 1, 2, \dots, N$ :

$$y_i^{(t)} = \min \{(x_i^{(t)} - x_i^{(L)}), (x_i^{(U)} - x_i^{(t)})\} / (x_i^{(U)} - x_i^{(L)}).$$

Arrange  $y_i^{(t)}$  in descending order of magnitude. Choose the first  $K$  variables as basic variables. Calculate the matrices  $J$  and  $C$  from the first derivative of constraints. Compute the reduced gradient  $\nabla \tilde{f}(x^{(t)})$  using Equation (4.46).

**Step 3** If  $\|\nabla \tilde{f}\| \leq \epsilon$ , **Terminate**;

Else set

$$\bar{d} = \begin{cases} 0 & \text{if } \bar{x}_i = x_i^{(L)} \text{ and } (\nabla \tilde{f})_i > 0, \\ 0 & \text{if } \bar{x}_i = x_i^{(U)} \text{ and } (\nabla \tilde{f})_i < 0, \\ -(\nabla \tilde{f})_i & \text{otherwise.} \end{cases}$$

$$\hat{d} = -J^{-1}C\bar{d}, \quad \hat{d} \text{ is a column vector.}$$

$$d = \begin{pmatrix} \hat{d} \\ \bar{d} \end{pmatrix}.$$

**Step 4** Find  $\alpha^{(t)}$  such that  $f(x^{(t)} + \alpha d)$  is minimum.

Set  $x^{(t+1)} = x^{(t)} + \alpha^{(t)}d$ , increment  $t$ , and go to Step 2.

As discussed earlier, this basic algorithm may create an infeasible solution at Step 4. In this case, an iterative procedure is to be used to find a feasible point satisfying all constraints.

Let us define an initial step length  $\alpha^0$ , a factor  $\gamma$  satisfying  $0 < \gamma < 1$ , and three termination parameters  $\epsilon_1, \epsilon_2, \epsilon_3$ . Then, Step 4 of the above algorithm can be replaced by the following step to handle nonlinear inequality constraints:

### Algorithm (contd.)

**Step 4 (revised)** Set  $\alpha = \alpha^0$ . Set  $i = 1$ .

- (a) Calculate  $v^{(i)} = x^{(t)} + \alpha d$ . If  $|h_k(v^{(i)})| \leq \epsilon_2$  for  $k = 1, 2, \dots, K$ , go to Step 4(d);  
 Else continue.
- (b) Let  $\hat{v}^{(i+1)} = \hat{v}^{(i)} - J^{-1}(v^{(i)}) \cdot h(v^{(i)})$  and  $\bar{v}^{(i+1)} = \bar{v}^{(i)}$ .
- (c) If  $\|\hat{v}^{(i+1)} - \hat{v}^{(i)}\| > \epsilon_3$ , set  $i = i + 1$ , and go to Step 4(b);  
 Else if  $|h_k(v^{(i+1)})| \leq \epsilon_2$  for  $k = 1, 2, \dots, K$ , go to Step 3(d);  
 Else set  $\alpha = \gamma\alpha$  and go to Step 4(a).
- (d) If  $f(x^{(t)}) \leq f(v^{(i)})$  set  $\alpha = \gamma\alpha$ ,  $i = 1$ , and go to Step 4(a);  
 Else set  $x^{(t+1)} = v^{(i)}$ ,  $t = t + 1$ , and go to Step 2.

### EXERCISE 4.9.1

We consider the constrained Himmelblau function to illustrate the working of the above algorithm.

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$\begin{aligned} g_1(x) &= 26 - (x_1 - 5)^2 - x_2^2 \geq 0, \\ g_2(x) &= 20 - 4x_1 - x_2 \geq 0, \\ x_1, x_2 &\geq 0. \end{aligned}$$

Here, both constraints are inequality constraints. Thus, we introduce two slack variables ( $x_3$  and  $x_4$ ) to transform them to equality constraints. This strategy of handling inequality constraints is known as the slack variable strategy. Thus, the constrained Himmelblau's problem is modified into the following problem:

$$\text{Minimize } (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$\begin{aligned} h_1(x) &= 26 - (x_1 - 5)^2 - x_2^2 - x_3 = 0, \\ h_2(x) &= 20 - 4x_1 - x_2 - x_4 = 0, \\ x_1, x_2, x_3, x_4 &\geq 0. \end{aligned}$$

In this problem, there are four decision variables and two constraints (that is  $N = 4$  and  $K = 2$ ). Therefore, there must be two basic and two nonbasic variables.

**Step 1** We choose an initial point  $x_1^{(0)} = 1$  and  $x_2^{(0)} = 2$ . Since we require an initial feasible solution, we find two other variables so that both constraints are satisfied. We obtain the initial point  $x^{(0)} = (1, 2, 6, 14)^T$ . We choose all termination parameters equal to  $10^{-3}$  and set the iteration counter  $t = 0$ .

**Step 2** At this step, we determine which two variables would be best for basic variables. Since the upper limits on neither of the variables  $x_1$  and  $x_2$  is specified, we assume some bounds. Let us assume that variables  $x_1$  and  $x_2$  lie in the domain  $(0, 5)$ . In this range, the minimum and maximum values of variable  $x_3$  computed from the first constraint are 1 and 26, respectively. For variables  $x_4$  they are  $-5$  and 20, respectively, as obtained from the second constraint. Thus, we compute the vector  $y = \{0.2, 0.4, 0.2, 0.24\}$ . We choose variables  $x_2$  and  $x_4$  as basic variables, since they correspond to larger values in the vector  $y$ . Thus, the first and third variables are chosen as nonbasic variables. Next, we compute the gradient of the constraints to form  $J$  and  $C$  matrices. By using numerical differentiation (Equation (3.4)), we obtain the gradient of the constraints at the initial point:  $\nabla h_1(x^{(0)}) = (8, -4, -1, 0)^T$  and  $\nabla h_2(x^{(0)}) = (-4, -1, 0, -1)^T$ . The matrix  $J$  is formed with the basic variables as rows and the matrix  $C$  is formed with the nonbasic variables as rows:

$$J = \begin{pmatrix} -4 & 0 \\ -1 & -1 \end{pmatrix}, \quad C = \begin{pmatrix} 8 & -1 \\ -4 & 0 \end{pmatrix}.$$

The first-order derivative of the objective function is computed numerically at the point  $x^{(0)}$ :  $\nabla f(x^{(0)}) = (-36, -32, 0, 0)^T$ . Thus, the basic and nonbasic vectors of this gradients are  $\widehat{\nabla f} = (-32, 0)$  and  $\bar{\nabla f} = (-36, 0)$ , respectively. Therefore, we compute the reduced gradient by using Equation (4.46) and by calculating the inverse of the matrix  $J$ :

$$\begin{aligned} \widetilde{\nabla f}(x^{(0)}) &= (-36, 0) - (-32, 0) \begin{pmatrix} -0.25 & 0 \\ 0.25 & -1 \end{pmatrix} \begin{pmatrix} 8 & -1 \\ -4 & 0 \end{pmatrix} \\ &= (-100, 8). \end{aligned}$$

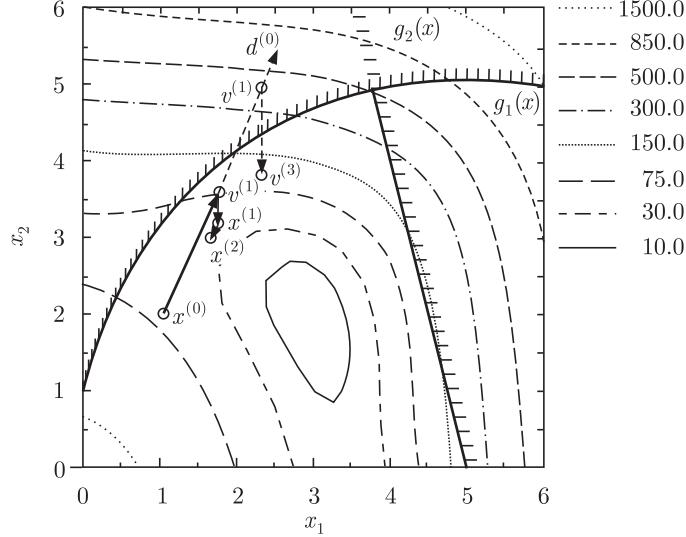
The above vector is a  $1 \times (N - K)$  or  $1 \times 2$  row vector.

**Step 3** Since the magnitude of the reduced gradient is not small, we compute the direction vector  $d^{(0)}$ . The nonbasic component of the direction vector  $\bar{d}$  is computed first. Since none of the nonbasic variables are at their boundaries at the current point, we assign  $\bar{d} = (-\nabla f)^T = (100, -8)^T$ . The vector  $\bar{d}$  is a  $(N - K) \times 1$  column vector and the vector  $\widehat{d}$  is a  $K \times 1$  column vector, computed as follows:  $\widehat{d} = -J^{-1}C\bar{d} = (202, -602)^T$ . Thus, the overall direction vector is  $d = (100, 202, -8, -602)^T$ . At this stage, we use the modified version of Step 4, because one of the constraints is nonlinear and the new point generated along this direction after a unidirectional search may not be feasible. In order to reduce the computational effort in each step, we follow the modified Step 4.

**Step 4** We first set a step factor  $\alpha = 0.015$ . Thereafter, we set  $i = 1$ .

**(a)** The new point is  $v^{(1)} = x^{(0)} + 0.015d$  and is found to be the point  $(2.500, 5.030, 5.880, 4.970)^T$ . The constraint values are  $h_1(v^{(1)}) = -11.430$  and  $h_2(v^{(1)}) = 0.000$ . The linear constraint is always satisfied.

Since all constraint violations are not small, we go to Step (b). The points  $x^{(0)}$  and  $v^{(1)}$  and the search direction  $d^{(0)}$  are shown in Figure 4.28.



**Figure 4.28** Intermediate points in two iterations of the generalized reduced gradient method. Infeasible points are made feasible by successive iterations of the Newton-Raphson method.

**(b)** The new basic variables are calculated using Equation (4.47)  $\hat{v}^{(2)} = \hat{v}^{(1)} - J^{-1}(v^{(1)}) \cdot h(v^{(1)})$ . Before we compute the new point, we calculate the matrix  $J$  at the current point  $v^{(1)}$  and then compute the inverse of the matrix. The new  $J$  matrix and its inverse are as follows:

$$J(v^{(1)}) = \begin{pmatrix} -10.060 & 0.000 \\ -1.000 & -1.000 \end{pmatrix}$$

$$J^{-1}(v^{(1)}) = \frac{1}{10.060} \begin{pmatrix} -1.000 & 0.000 \\ 1.000 & -10.060 \end{pmatrix}.$$

Thus, the new point is calculated as follows:

$$\begin{aligned} \hat{v}^{(2)} &= \begin{pmatrix} 5.030 \\ 4.970 \end{pmatrix} - \frac{1}{10.060} \begin{pmatrix} -1.000 & 0.000 \\ 1.000 & -10.060 \end{pmatrix} \cdot \begin{pmatrix} -11.430 \\ 0.000 \end{pmatrix}, \\ &= \begin{pmatrix} 3.894 \\ 6.106 \end{pmatrix}. \end{aligned}$$

The nonbasic variables are the same as before:  $\bar{v}^{(2)} = (2.500, 5.880)^T$ . Using the above transformation, only basic variables are modified so that along with nonbasic variables they satisfy both equality constraints.

**(c)** The two points  $v^{(1)}$  and  $v^{(2)}$  are not close to each other, thus we increment the counter  $i = 2$  and go to Step (b).

**(b)** Another set of basic variables is computed by calculating the  $J$  matrix at point  $v^{(2)}$ :

$$\begin{aligned}\hat{v}^{(2)} &= \begin{pmatrix} 3.894 \\ 6.106 \end{pmatrix} - \frac{1}{7.788} \begin{pmatrix} -1.000 & 0.000 \\ 1.000 & -7.788 \end{pmatrix} \cdot \begin{pmatrix} -1.293 \\ 0.000 \end{pmatrix}, \\ &= \begin{pmatrix} 3.728 \\ 6.272 \end{pmatrix}.\end{aligned}$$

The nonbasic variables remain the same as before. Thus the new point is  $v^{(3)} = (2.500, 3.728, 5.880, 6.272)^T$  with a violation of the first constraint of only  $-0.028$ . Figure 4.28 shows how the infeasible point  $v^{(1)}$  is made feasible. Since the variable  $x_2$  is a basic variable and the variable  $x_1$  is a nonbasic variable, the infeasible point is only modified on the variable  $x_2$  to make it feasible. One may wonder as to why the Newton-Raphson method could not find a point on the constraint  $g_1(x)$ . Recall that the original problem has now been changed to a four-variable problem. A plot of only two variables does not reveal the true scenario. The point  $v^{(3)}$  actually lies close to the intersection of the constraint boundaries in four variables. At the end of this step, we move to Step (d).

**(d)** The function value at the initial point is  $f(x^{(0)}) = 68$  and at the current point is  $f(v^{(3)}) = 88.81$ , which is worse than the initial point. Thus, we reduce the step parameter:  $\alpha = 0.5\alpha^0 = 0.0075$ . We set  $i = 1$  and go to Step (a).

**(a)** The new point is  $v^{(1)} = x^{(0)} + \alpha d = (1.750, 3.515, 5.940, 9.485)^T$ . The constraint violations are  $h_1(v^{(1)}) = -2.875$  and  $h_2(v^{(1)}) = 0.0$ . The point  $v^{(1)}$  is also shown in Figure 4.28, close to the constraint  $g_1(x)$ . The figure shows that the point is feasible as far as the original two constraints  $g_1(x)$  and  $g_2(x)$  (in two-variables) are concerned. But this point is not feasible for the two modified equality constraints  $h_1(x)$  and  $h_2(x)$ . Computing the  $J$  matrix at this point, we find the new point  $v^{(2)} = (1.750, 3.108, 5.940, 9.892)^T$ . The violation of the first constraint reduces to a value  $-0.162$ . One more iteration yields the point

$$v^{(3)} = (1.750, 3.082, 5.940, 9.918)^T.$$

At this point, the constraint violation is below the permissible limit. Thus, we move to Step (d) and compare the function values.

**(d)** The function value at the point  $v^{(3)}$  is  $f(v^{(3)}) = 41.627$ , which is less than that at  $x^{(0)}$ . Thus, we accept this point  $x^{(1)} = (1.750, 3.082, 5.940, 9.918)^T$  and move to Step 2. This completes one iteration of the generalized reduced gradient method.

**Step 2** By calculating the  $y$ -vector, we obtain the basic and nonbasic variables as follows:  $\hat{x} = (x_2, x_4)^T$  and  $\bar{x} = (x_1, x_3)^T$ . The matrices  $J$  and  $C$  are computed at the point  $x^{(1)}$  and the reduced gradient vector is computed:

$$\begin{aligned}\nabla \tilde{f}(x^{(1)}) &= (-25.491, 0.000) - (42.667, 0.000) \begin{pmatrix} -6.164 & 0.000 \\ -1.000 & -1.000 \end{pmatrix}^{-1} \\ &\times \begin{pmatrix} 6.500 & -1.000 \\ -4.000 & 0.000 \end{pmatrix} = (19.502, -6.922).\end{aligned}$$

**Step 3** The direction vector is found to be  $d = (-19.502, -21.703, 6.922, 99.780)^T$ .

**Step 4** At the starting point with  $\alpha = 0.0075$ , we obtain the feasible point  $v^{(1)} = (1.604, 2.919, 5.992, 10.673)^T$  with a constraint violation of  $h_1(v^{(1)}) = -0.045$  only. After one iteration of Step 4, we obtain the point  $v^{(2)} = (1.604, 2.919, 5.992, 10.673)^T$  with a function value equal to 39.90. Since there is an improvement in the function value, the point  $v^{(2)}$  is renamed as  $x^{(2)}$  and we continue with Step 2.

The algorithm proceeds this way until a small value of the reduced gradient is found. Notice that the magnitude of the reduced gradient  $\nabla \tilde{f}$  gets smaller at every iteration.

## 4.10 Gradient Projection Method

Similar to the reduced gradient method, the gradient projection method is also suitable for solving NLP problems having equality constraints only. As we have discussed before, one difficulty with the reduced gradient method is that at every iteration two matrices are required to be formed and the inverse of a  $(K \times K)$  matrix is required to be computed. In certain cases, the chosen basic variables may be such that the matrix  $J$  becomes singular, which will result in a premature termination of the algorithm. In the gradient projection method, partitioning of the constraint gradient into two matrices is avoided and an intermediate infeasible point is made feasible by projecting the point onto the constraint surface. Like the reduced gradient method, inequality constraints can be taken care of either by adding a slack variable for each inequality constraint or by using an active constraint strategy. In the algorithm discussed here, the latter is used.

At every iteration, the active constraint set (comprising constraints having a value equal to zero at the current point) is found. If no constraints are active, the steepest descent search method is adopted. Otherwise, the steepest descent search direction is first found and then projected onto the intersecting surface of the active constraints to create a feasible search direction. If the magnitude of the resulting search direction vector is smaller

than a specified small number, either the optimum is reached or the active constraints at the current point are orthogonal to each other. In the latter case, one active constraint is excluded from the constraint set and a new search direction is found. Once a feasible search direction is found, a unidirectional search is performed to obtain the minimum point along that direction. The difficulty with this technique is that in the case of nonlinear constraints, the intermediate points in the unidirectional search may not fall on the constraint boundaries. Thus, in order to make the search points feasible, the unidirectional search method is modified. Every point created in the search is projected onto the intersection surface of the constraints in order to make the point feasible. Since this may require a number of function evaluations, only three feasible points are chosen along the search direction and the quadratic interpolation search method described in Chapter 2 is used to obtain a guess for the optimum point along that direction. The search is continued from the new point. This process continues until no better solution could be found.

In the following, we describe the algorithm and then show hand-calculations of a few iterations of this algorithm on the constrained Himmelblau's function.

### Algorithm

**Step 1** Choose an initial point  $x^{(0)}$ , termination factors  $\epsilon_1$  and  $\epsilon_2$ , and an iteration counter  $t = 0$ .

**Step 2** Evaluate all inequality constraints at  $x^{(t)}$  to identify the active set

$$I^{(t)} = \{j : |g_j(x^{(t)})| \leq \epsilon_1, j = 1, 2, \dots, J\}.$$

**Step 3** If  $t \geq 1$  and  $I^{(t)} = I^{(t-1)}$  or  $I^{(t)} = \emptyset$  (conditions for *steepest descent* search), set  $s = -\nabla f(x^{(t)})$  and go to Step 4;

Else form the matrix  $A$  whose rows are  $\nabla h_k$  and  $\nabla g_j$ , where  $j \in I^{(t)}$ . Calculate the projection matrix  $P = I - A^T(AA^T)^{-1}A$  and the search direction  $s = -P\nabla f(x^{(t)})$ .

**Step 4** If  $\|s^{(t)}\| > \epsilon_2$ , go to Step 5;

Else calculate constraint multipliers

$$(v, u) = (AA^T)^{-1}A\nabla f$$

and find  $u_m = \min\{u_\ell : \ell \in I^{(t)}\}$ . If  $|u_m| \leq \epsilon_2$ , **Terminate**;

Else delete inequality constraint  $m$  from  $I^{(t)}$  and go to Step 3.

**Step 5** Determine the maximum step size  $\alpha_{\max}$  such that  $g_\ell(w(\alpha)) \geq 0$  for all  $\ell \notin I^{(t)}$ , where any point along  $s^{(t)}$  is represented by  $w(\alpha)$ .

**Step 6** If the *steepest descent* search is to be performed, perform a unidirectional search in the range  $(0, \alpha_{\max})$  to calculate  $\alpha^*$ ;

Else conduct a bracketing search on  $\alpha$  in the interval  $(0, \alpha_{\max})$  to determine three points  $w(\alpha_1)$ ,  $w(\alpha_2)$ , and  $w(\alpha_3)$  which bracket the minimum point of  $f(x)$  along the intersecting surface of equality and active inequality constraints. For each  $\alpha$ ,  $w(\alpha)$  is the final solution of following iterations on the active constraints:

$$w^{(t+1)} = w^{(t)} - A^T (A A^T)^{-1} H.$$

Calculate the matrix  $A$  at the point  $w^{(t)}$  and the vector  $H$  is the constraint values of all equality and active inequality constraints at the point  $w^{(t)}$ . Then use a quadratic interpolation to estimate  $\alpha^*$ .

**Step 7** Set  $x^{(t+1)} = w(\alpha^*)$ ,  $t = t + 1$ , and go to Step 2.

The difficulty with this method is that the matrix  $(A A^T)^{-1}$  is evaluated at every iteration and every new infeasible point needs to be projected on the constraint surface. If the optimum lies on only one of the constraints, this method makes the search faster.

#### EXERCISE 4.10.1

We illustrate the working of this algorithm on the constrained Himmelblau function.

$$\text{Minimize } (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

$$g_3(x) = x_1 \geq 0,$$

$$g_4(x) = x_2 \geq 0.$$

**Step 1** We choose an initial point  $x^{(0)} = (0, 0)^T$ , all termination factors  $\epsilon_1 = \epsilon_2 = 10^{-3}$ , and set the iteration counter  $t = 0$ .

**Step 2** At the initial point, two constraints ( $g_3$  and  $g_4$ ) are active. Thus,  $I^{(0)} = \{3, 4\}$ .

**Step 3** At this step, we form the matrix  $A$  from the gradient of the active constraints:  $g_3(x^{(0)}) = (1, 0)^T$  and  $g_4(x^{(0)}) = (0, 1)^T$ . Thus the matrix  $A$  is formed as follows:

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

which is an identity matrix. We observe that the matrix  $(A A^T)^{-1}$  is also an identity matrix. We now compute the projection matrix as follows:

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

The gradient vector of the objective function at the current point is  $\nabla f(x^{(0)}) = (-14, -22)^T$ . Thus, the search direction is  $s = P \nabla f(x^{(0)}) = (0, 0)^T$ , a zero vector.

**Step 4** Since  $\|s\| = 0$ , we calculate the constraint multipliers:

$$u = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -14 \\ -22 \end{pmatrix} = \begin{pmatrix} -14 \\ -22 \end{pmatrix}.$$

Since  $u_2$  is most negative, the constraint  $g_4(x)$  is excluded from the active constraint set. Thus, we update the set  $I^{(0)} = \{3\}$  and we move to Step 3. The set  $I^{(0)}$  is not empty; we construct a new  $A$  matrix.

**Step 3** The matrix  $A$  now has only one row:  $A = (1, 0)$ . The projection matrix is calculated as

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

The search direction is computed by multiplying the negative of the gradient vector  $\nabla f(x^{(0)})$  with this projection matrix:  $s = (0, 22)^T$ .

**Step 4** The norm of the search vector  $s$  is not small. Thus, we move to Step 5.

**Step 5** At this step, we have to determine the maximum permissible step size for the other three constraints. We find a generic point along the search direction  $s$  as

$$w(\alpha) = x^{(0)} + \alpha s = (0, 22\alpha)^T.$$

For the first constraint, the maximum permissible  $\alpha$  can be calculated by solving  $g_1(w(\alpha)) = 0$ . This is a root-finding problem which can be formulated as an optimization problem described in Section 2.6. Using that technique, we obtain the solution  $\alpha_1 = 0.045$ . Similarly, for the second and fourth constraints, the corresponding limits are  $\alpha_2 = 0.909$  and  $\alpha_4 = 0$ , respectively. Since  $\alpha_4$  is zero, we find the minimum of only other two values and obtain  $\alpha_{\max} = 0.045$ .

**Step 6** Thus, the bounds of the search parameter  $\alpha$  are 0 and 0.045, respectively. We take three points along the search direction to estimate the minimum point along that direction. Let us say the chosen values for  $\alpha$  are  $\alpha_1 = 0.000$ ,  $\alpha_2 = 0.020$  and  $\alpha_3 = 0.045$ . We observe that the corresponding points  $(w(\alpha))$  are  $w1^{(0)} = (0.00, 0.00)^T$ ,  $w2^{(0)} = (0.000, 0.440)^T$ , and  $w3^{(0)} = (0.000, 1.000)^T$ , respectively. Knowing that the constraint  $g_3(x)$  is the only active constraint being considered in this iteration, we check each of these points for feasibility. Since the active constraint  $g_3(x)$  is linear, all points along the projected search direction  $s$  lie on the constraint. This can be verified by projecting any of the above points onto the constraint surface  $g_3(x)$ . Knowing the matrices

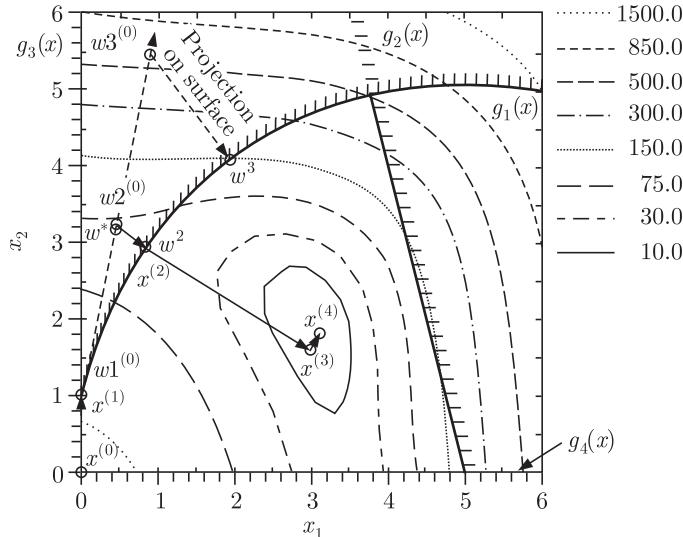
$$A = (1, 0), \quad H = (0.000)^T,$$

we project the second point  $w2^{(0)}$  onto the constraint surface  $g_3(x)$ :

$$\begin{aligned} w2^{(1)} &= w2^{(0)} - A^T(AA^T)^{-1}H, \\ &= \begin{pmatrix} 0.000 \\ 0.440 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} \left( (1, 0) \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right)^{-1} (0.000), \\ &= \begin{pmatrix} 0.000 \\ 0.440 \end{pmatrix}. \end{aligned}$$

Thus, we get back the same point  $w2^{(0)}$ . Similarly, other points can also be shown to lie on  $g_3(x)$  after projection. The function values at these points are  $f(w1) = 170.000$ ,  $f(w2) = 157.841$ , and  $f(w3) = 136.000$ . With these function values, we now have to estimate the minimum point along  $s$  in the range  $\alpha \in (0, 0.045)$ . We observe that the function monotonically reduces with the increase in  $\alpha$ . Assuming that there is only one minimum point along  $s$ , we conclude that the minimum lies at  $\alpha^* = 0.045$  or at the point  $w3^{(0)}$ .

**Step 7** The corresponding point is  $x^{(1)} = (0, 1)^T$ . We set  $t = 1$  and go to Step 2. This completes one iteration of the gradient projection method. The progress of the algorithm is shown in Figure 4.29.



**Figure 4.29** Intermediate points in a number of iterations of the gradient projection method. Infeasible points are projected onto the active constraint surface.

**Step 2** At this step, there are two active constraints:  $I^{(1)} = \{1, 3\}$ , as seen from Figure 4.29.

**Step 3** We form the matrix  $A$  by computing the gradient of the active constraints at  $x^{(1)}$ :

$$A = \begin{pmatrix} 10 & -2 \\ 1 & 0 \end{pmatrix}.$$

The corresponding projection matrix is found to be a zero matrix. Thus, the search direction is also a zero vector.

**Step 4** Since the search direction is a zero vector, we calculate the constraint multipliers. The gradient of the objective function at this point is

$$\nabla f(x^{(1)}) = (-12, -44)^T.$$

The constraint multipliers are  $u = (22, -232)^T$ . Thus, we eliminate the third constraint from the active set and recalculate the projection matrix at Step 3.

**Step 3** The matrix  $A$  is now  $A = (10, -2)$  and the corresponding projection matrix is

$$P = \begin{pmatrix} 0.029 & 0.192 \\ 0.192 & 0.962 \end{pmatrix}.$$

The search vector is  $s = (8.922, 44.613)^T$ , which is shown in Figure 4.29 with a dashed arrow from  $x^{(1)}$ .

**Step 4** The search vector is not small, so we move to Step 5.

**Step 5** Like in the previous iteration, we compute the upper bounds for the second, third, and fourth constraints:  $\alpha_2 = 0.237$ ,  $\alpha_3 = 0.000$ , and  $\alpha = -0.020$ . The negative value of  $\alpha_4$  signifies that the constraint  $g_4(x)$  can be found in the negative  $s$  direction. The latter two values are not accepted; thus we set  $\alpha_{\max} = 0.237$ .

**Step 6** Let us again choose three different  $\alpha$  values:  $\alpha_1 = 0.000$ ,  $\alpha_2 = 0.050$  and  $\alpha_3 = 0.100$  (all are within the limits). The corresponding points are  $w1^{(0)} = (0, 1)^T$ ,  $w2^{(0)} = (0.446, 3.231)^T$ , and  $w3^{(0)} = (0.892, 5.461)^T$ , respectively. Since,  $g_1(x)$  is not linear, these points may not fall on the constraint surface. In order to investigate whether they bracket the minimum, we first project each of these points onto the constraint surface to find a corresponding feasible point and then compare the slopes at these points. Of course, the first point is on the surface. The second point does not fall on the surface because  $g_1(w2^{(0)}) = -5.178$ . This point can be made to fall on the surface by calculating the  $A$  matrix at the point and by performing the following modification:

$$\begin{aligned}
w2^{(1)} &= w2^{(0)} - A^T(AA^T)^{-1}H = \begin{pmatrix} 0.446 \\ 3.231 \end{pmatrix} \\
&- \begin{pmatrix} 9.108 \\ -6.462 \end{pmatrix} \left( (9.108, -6.462) \begin{pmatrix} 9.108 \\ -6.462 \end{pmatrix} \right)^{-1} (-5.178), \\
&= \begin{pmatrix} 0.824 \\ 2.963 \end{pmatrix}.
\end{aligned}$$

At this point, the constraint violation is  $g_1(w2^{(1)}) = -0.218$ , an improvement from the point  $w2^{(0)}$ . One more iteration yields the point  $w2^{(2)} = (0.841, 2.951)^T$  with a constraint violation of only  $g_1(w2^{(2)}) = -0.005$ . Similarly, the third point ( $g_1(w3^{(0)}) = -20.698$ ) can also be made to fall on the surface and the new point is  $w3^{(2)} = (1.931, 4.071)^T$  with constraint violation of only 0.002. Thus, the projected points along the constraint  $g_1(x)$  are  $w1 = (0.000, 1.000)^T$ ,  $w2 = (0.841, 2.951)^T$ , and  $w3 = (1.931, 4.071)^T$ . Now we compute the derivatives at the extreme points ( $w1$  and  $w3$ ) to see whether these points bracket the minimum. The derivative of the objective function with respect to  $\alpha$  at the points corresponding to  $\alpha_1$  and  $\alpha_3$  are  $\partial f / \partial \alpha|_{\alpha=0} = -2164.0$  and  $\partial f / \partial \alpha|_{\alpha=0.1} = 755.0$ , respectively. Since they are opposite in sign, these two points bracket the optimum. Now, we compute the function values at these points  $f(w1) = 136.000$ ,  $f(w2) = 60.400$ , and  $f(w3) = 143.780$ . Using the quadratic interpolation on these values (refer to Section 2.4), we obtain the minimum point  $\alpha^* = 0.049$ . The corresponding point is  $w^* = (0.435, 3.177)^T$ . Since the constraint violation at this point is  $g_1(w^*) = -4.932$ , we project this point onto the constraint surface  $g_1(x)$  and obtain the new feasible point  $w^* = (0.815, 2.913)^T$ .

**Step 7** The new point is  $x^{(2)} = (0.815, 2.913)^T$  with a function value  $f(x^{(2)}) = 60.395$ . In order to continue, we have to set  $t = 2$  and proceed to Step 2.

At the third iteration, an interesting scenario happens. At the new point, only the third constraint is active again. Thus, the search will continue along the tangent at the point  $x^{(2)}$  on the constraint  $g_1(x)$  and we converge to the same point. Step 3 of the algorithm takes care of this situation. When the active constraint set  $I^{(t)}$  at any iteration  $t$  is the same as that in iteration  $(t-1)$ , we cannot improve the point any further along that constraint surface. Thus, we use the steepest descent search method to investigate whether there is any better point away from the constraint surface. If a better point is found, we continue to use the gradient projection algorithm; otherwise the current point is the optimum point. At the point  $x^{(2)}$ , we set the search direction equal to the negative of the objective function  $s = (19.597, -11.971)^T$  and perform a unidirectional search. We obtain the minimum point  $x^{(3)} = (3.031, 1.559)^T$  with a function value equal to  $f(x^{(3)}) = 2.436$ . Since no constraints are active

at this point, we again search along the steepest descent direction, and obtain the next point  $x^{(4)} = (3.148, 1.752)^T$  with a function value  $f(x^{(4)}) = 1.054$ . Continuing this process further will lead to the optimum solution. Figure 4.29 shows the progress of these iterations. In general, the search process continues along the boundary of the feasible region. If the optimum point is on the boundary, the optimum is eventually found by traversing from one boundary to another. On the other hand, if the optimum is inside the feasible region, the search deviates from the boundary after some iterations and follows the steepest descent directions through the feasible region.

Although the active constraint strategy is used in this exercise problem, the slack variable strategy could also be used to handle inequality constraints. Similarly, the active constraint strategy can also be used equally efficiently in the reduced gradient method. The active constraint strategy requires a book-keeping of active constraints at each iteration, but the slack variable strategy usually requires more computational time owing to the use of a larger number of variables. Thus, for a few inequality constraints, the slack variable strategy may be efficient; otherwise active constraint strategy can be used.

## 4.11 Summary

In this chapter, we have presented a number of nonlinear programming methods for constrained optimization. To begin with, the Kuhn-Tucker conditions for optimality have been discussed. Kuhn-Tucker conditions for constrained optimization problems are derived on the basis of unconstrained optimization of the corresponding Lagrange function. Kuhn-Tucker points are those points which satisfy all Kuhn-Tucker conditions. It turns out that Kuhn-Tucker points are likely candidates for constrained optimal points. The Kuhn-Tucker necessity theorem helps identify non-optimality of a point. Although the Kuhn-Tucker sufficiency theorem helps to identify the optimality of a point, the theorem can only be applied for a limited class of constrained NLP problems.

Optimization algorithms described in this chapter are divided into two broad categories—direct search methods and gradient-based methods. Among the direct search methods, the penalty function method is most widely used. In the penalty function method, infeasible solutions are penalized by adding a penalty term in relation to the amount of the constraint violation at that solution. There are primarily two types of penalty functions—interior penalty functions which penalize only feasible solutions close to constraint boundaries and exterior penalty functions which penalize infeasible points. The interior penalty methods require an initial feasible solution, whereas the exterior penalty methods do not require the point to be feasible. This is why exterior penalty methods are more popular than interior penalty methods. The bracket-operator exterior penalty function method has been applied to a constrained Himmelblau function. The results have shown that even though the penalty function method solves the above function to optimality, it distorts

the objective function in successive iterations. This problem of distortion of the objective function can be rectified by using a method of multiplier (MOM) strategy. In the MOM technique, the penalty is not added directly to the objective function, instead, some function of the penalty is added. The effect is a shift of the unconstrained objective function near the constrained optimum point. One added advantage of the MOM technique is that the final iteration statistics can be directly used to compute the Lagrange multipliers corresponding to constraints.

Lagrange multipliers are important for performing *sensitivity analysis* of a constrained problem. Sensitivity analysis is usually performed to investigate the change in the optimal objective function value due to a change in the constraint parameters. It turns out that this quantity is linearly proportional to the Lagrange multipliers corresponding to constraints. A hand-calculation has been shown to perform the sensitivity analysis on an exercise problem.

In a constrained optimization problem, equality constraints make the search process slow and difficult to converge. The best way to handle equality constraints is to use them to eliminate some variables, if possible. Even if explicit elimination of a variable is not evident, one variable can be theoretically eliminated using one equality constraint by the root-finding method suggested in Chapter 2. A simple version of the variable elimination method has been presented first. In a later section, a more sophisticated variable elimination method has been discussed.

Among the other direct search methods, the complex search method and an adaptive random search method are discussed. The complex search method for constrained optimization problems is similar to the simplex search method for unconstrained functions, except that the provisions are made to check the feasibility of the intermediate solutions. If solutions are found to be infeasible, suitable modifications are adopted to make them feasible. If the constrained optimum lies inside the feasible region, this method is more efficient. Random search methods create a set of random points in the feasible region and compare them to find the best point. The search space is reduced at each iteration centring around the best point found in the previous iteration. These techniques may be useful in problems where structured optimization methods fail to find a reasonable solution or where the search space is too narrow. Often, random search methods are used to find a feasible point in the search space. Thereafter, a structured optimization method (a gradient search method or a sophisticated direct search method) can be invoked from that feasible point.

Since linear programming (LP) techniques are simple and have convergence proofs, a number of NLP methods have been structured to use LP techniques iteratively. In this chapter, we have discussed two such methods. The Frank-Wolfe algorithm linearizes the objective function and constraints at any point and solves the resulting LP problem using a LP technique. A unidirectional search is then performed from the old point to the new point to find a better point. The objective function and constraints are linearized at this point and the search continues. Although this method is simple and

straightforward, it may not work if the initial point is far away from the true optimum point, because the linearized constraint and the objective function may be very different than the true functions at a point away from the current point. The cutting plane method begins with a large search space formed by linear constraints. The resulting LP problem is solved and a linear cutting plane is formed using the most violated constraint at the resulting point. This is how cutting planes are formed at each iteration to reduce the initial large search space into the shape of the true feasible region. It has been proved elsewhere (Kelly, 1960) that for convex objective functions and feasible regions, the cutting planes do not exclude any part of the true feasible region. One way to handle nonlinear objective functions is also discussed. Since the cutting plane method always surrounds the true feasible search space with linear planes and the solution is obtained by using those linear planes as constraints, the obtained solution is always infeasible. Thus, sufficiently large number of iterations are required so that the solution, although infeasible, is very close to the true optimum solution. This method is not very efficient if the true optimum point is not a boundary point. A local exhaustive search can then be performed to find the true optimum solution.

Next, we have described quadratic programming (QP) algorithm which makes a quadratic approximation of the objective function and a linear approximation of all constraints. It turns out that the resulting problem can be solved using the LP technique. The sequential quadratic programming (SQP) method has been described thereafter.

The feasible direction method starts its search along a feasible direction from an initial feasible point. At each point, the active constraints are first found. Thereafter, a search direction is found by maximally satisfying the descent and feasibility properties of a direction. This requires solving a linear programming problem. If the point does not make any constraint active, the steepest descent direction is used. When a search direction is found, a unidirectional search is adopted and a new point is found. This method is largely known as Zoutendijk's method. If the true optimum falls on a constraint boundary, this method may require a number of iterations before converging close to the optimum point.

The generalized reduced gradient method is a sophisticated version of the variable elimination method of handling equality constraints. Some variables (called basic variables) are expressed in terms of other variables (called nonbasic variables) by using the equality constraints. The gradient of the objective function at any point is then expressed in terms of only nonbasic variables. This gradient is called the reduced gradient. Thus, this method is similar in principle to the steepest descent method except that some variables are expressed in terms of other variables. Although a method to handle inequality constraints is discussed, this algorithm works very well for NLP problems with equality constraints only.

Another algorithm to handle equality constraints efficiently is to use the gradient projection method where each infeasible point, obtained by performing a unidirectional search, is projected on to the constraint surface to make the point feasible. Thus, the resulting points are always feasible. If two successive iterations produce the same point, the steepest descent method is adopted to check whether there is any better point inside the feasible region. An active constraint strategy has been used to illustrate the handling of inequality constraints.

## REFERENCES

- Bazaraa, M. S., Sherali, H. D., and Shetty, C. M. (2004). *Nonlinear Programming: Theory and Algorithms*. Singapore: Wiley.
- Bector, C. R., Chandra, S. and Dutta, J. (2005). *Principles of Optimization Theory*. New Delhi: Narosa Publishing House.
- Box, M. J. (1965). A new method of constrained optimization and a comparison with other methods. *Computer Journal*, **8**, 42–52.
- Kelly, J. E. (1960). The cutting plane method for solving convex programs. *SIAM Journal*, **8**, 703–712.
- Luus, R. and Jaakola, T. H. I. (1973). Optimization by direct search and systematic reduction of the size of search region. *AICHE Journal*, **19**, 760–766.
- Mangasarian, O. L. (1969). *Nonlinear Programming*. New York: McGraw-Hill.
- Rao, S. S. (1993). Genetic algorithmic approach for multiobjective optimization of structures. In *Proceedings of the ASME Annual Winter Meeting on Structures and Controls Optimization*, volume 38, pages 29–38.
- Rao, S. S. (1984). *Optimization Theory and Applications*. New Delhi: Wiley Eastern.
- Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. (1983). *Engineering Optimization—Methods and Applications*. New York: Wiley.
- Rockafellar, R. T. (1996). *Convex Analysis*. Princeton: Princeton University Press.
- Strang, G. (1980). *Linear Algebra and Its Applications*. Orlando: Academic Press.
- Taha, H. A. (1989). *Operations Research*. New York: MacMillan.
- Zangwill, W. I. (1969). *Nonlinear Programming*. Englewood Cliffs, New Jersey: Prentice-Hall.
- Zoutendijk, G. (1960). *Methods of Feasible Directions*. Amsterdam: Elsevier.

## PROBLEMS

**4-1** In the following minimization problem

$$\text{Minimize } x_1^2 + x_2$$

subject to

$$g_1(x) = 10 \exp(x_1^2 + x_2) - 5x_1 + 12 \geq 0,$$

$$g_2(x) = 5x_1^3 + 2x_2 - 9 \leq 0,$$

$$0 \leq x_1, x_2 \leq 3,$$

find whether the following points are feasible. If they are feasible, which of the above constraints are active?

- (i)  $(0, 1)^T$ .
- (ii)  $(1, 4)^T$ .
- (iii)  $(2, 1)^T$ .
- (iv)  $(3, 0)^T$ .

**4-2** Write down the Kuhn-Tucker conditions for the above NLP problem. Check whether the above points are K-T points.

**4-3** Consider the constrained optimization problem:

$$\text{Minimize } 10x_1^2 + 2.5x_2^2 - 5x_1x_2 - 1.5x_1 + 10$$

subject to

$$x_1^2 + 2x_2^2 + 2x_1 \leq 5.$$

Find whether any of the following points are likely candidates of the optimum point:

- (i)  $(0, 0)^T$ .
- (ii)  $(0.1, 0.1)^T$ .
- (iii)  $(2, 1)^T$ .

**4-4** Identify whether the points

- (i)  $(0, 6)^T$ ,
- (ii)  $(1.5, 1.5)^T$ ,
- (iii)  $(2, 2)^T$

are optimal points to the following NLP problem:

$$\text{Minimize } x_1^2 + x_2^2 - 10x_1 + 4x_2 + 2$$

subject to

$$x_1^2 + x_2 - 6 \leq 0,$$

$$x_2 \geq x_1,$$

$$x_1 \geq 0.$$

**4-5** Write down the Kuhn-Tucker conditions for the following problem:

$$\text{Maximize } 3x_1^2 - 2x_2$$

subject to

$$2x_1 + x_2 = 4,$$

$$x_1^2 + x_2^2 \leq 19.4,$$

$$x_1 \geq 0.$$

Find out whether points  $(0, 4)^T$  and  $(3.4, -2.8)^T$  are Kuhn-Tucker points. How would the maximum function value change if the equality constraint is changed to the following:  $2x_1 + x_2 = 6$ ?

**4-6** In an NLP problem, the following constraints are used:

$$g_1(x) = (x_1 - 5)^2 + x_2^2 - 25 \leq 0,$$

$$g_2(x) = x_1 + 2x_2 - 12 \leq 0,$$

$$g_3(x) = 2x_1 + x_2 + 4 \geq 0.$$

The current point is  $(0, 0)^T$  and a search direction  $\hat{d} = (2, 1)^T$  is found. Find the minimum and maximum allowable bounds along this direction.

**4-7** Consider the following constrained Himmelblau's function:

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$(x_1 - 5)^2 + x_2^2 \leq 26$$

$$4x_1 + x_2 \leq 20$$

$$x_1, x_2 \geq 0$$

- (a) Write the Kuhn-Tucker conditions.
- (b) Are the following points Kuhn-Tucker points?
  - (i)  $(0, 1)^T$
  - (ii)  $(3, 2)^T$

**4-8** Solve the following single-variable optimization problem using the geometric programming method:

$$\text{Minimize } f(x) = x^3 - x, \quad x > 0.$$

How does the solution change if a constraint  $x \geq 0.7$  is added? Form the primal and dual problem in each case and solve.

**4-9** For the problem

$$\text{Minimize } f(\mathbf{x}) = (x_1 - 3)^2 + (x_2 - 3)^2,$$

$$\text{subject to } 2x_1 + x_2 \leq 2,$$

develop the dual function, maximize it and find the corresponding point in  $\mathbf{x}$ -space.

**4-10** For the primal problem

$$\text{Minimize } (x_1 - 2)^2 + (x_2 - 1)^2,$$

subject to

$$x_1 + x_2 - 2 \leq 0,$$

$$x_1^2 - x_2 \leq 0,$$

construct the dual problem and find the dual solution.

**4-11** The primal problem is to minimize  $x^3$  such that  $x \geq 0$ . Find the dual solution.

**4-12** For the problem

$$\text{Maximize } f(x) = x^2, \quad \text{subject to } 0 \leq x \leq 1,$$

show that there exists an infinite duality gap for this problem (Bector, Chandra and Dutta, 2005).

**4-13** A thin right circular conical container (base diameter 10 cm and height 12.5 cm) is cut by a plane  $x + y + 1.5z = 10$  (the origin is assumed to be at the centre of the base circle). Find the point on the cut surface closest to the apex of the cylinder using the variable elimination method.

**4-14** Find the point on the ellipse defined by the intersection of the surfaces  $x + y = 1$  and  $x^2 + 2y^2 + z^2 = 1$  and is nearest to the origin. Use the Lagrange function method.

**4-15** The intersection of the planar surface  $x + y + 4z = 2$  and a cone  $x^2 + 2y^2 + z^2 = 1$  creates an ellipse.

- (i) Formulate an optimization problem to locate a point on the ellipse which is nearest to the origin.
- (ii) Write and solve the resulting KKT conditions to find the nearest point.

**4-16** Consider the NLP problem:

$$\text{Minimize } f(x) = 10 + x^2 - 8x$$

subject to

$$x \geq 6.$$

Use the penalty function method with the following penalty terms to find the constrained minimum point of the above problem:

- (i) Bracket operator penalty term with the following values of  $R$ : 0.01, 0.1, 1.0, 10.0, 100.0, and  $\infty$ .
- (ii) Inverse penalty term with following values of  $R$ : 1,000.0, 100.0, 10.0, 1.0, 0.1, and 0.01.

Form the penalized function and use the exact differentiation technique to compute the optimal point in each sequence.

**4-17** Consider the following optimization problem:

$$\text{Minimize } (x_1 - 1)^2 + (x_2 - 1)^2,$$

subject to

$$x_1 - x_2 - 2 = 0,$$

$$x_1 + x_2 - 0.5 \leq 0.$$

- (i) Solve the problem graphically.
- (ii) Use Augmented Lagrangian method to find the optimal  $\mathbf{x}$  vector at each iteration. Use the bracket operator ( $\langle g \rangle = g$ , if  $g < 0$ ; zero, otherwise) to handle the inequality constraint ( $g \geq 0$ ). Calculate  $\sigma$ ,  $\tau$ ,  $\mathbf{x}$ , and  $f$  for three iterations and show them on a table as shown below. Start with  $\sigma = \tau = 0$  and use penalty parameter  $R = 1$ .
- (iii) Using the steady-state termination condition ( $t = \infty$ ), find the optimal  $\sigma$ ,  $\tau$ ,  $\mathbf{x}$ , and  $f$  values.
- (iv) From the above optimal values, compute the Lagrange multipliers of both constraints.

$t$	$\sigma^{(t)}$	$\tau^{(t)}$	$x_1^{(t)}$	$x_2^{(t)}$	$f^{(t)}$
0					
1					
2					
3					
$\infty$					

**4-18** A rope of length  $P$  is to be used to form a regular polygon having  $n$  equal sides. It is required to form a polygon for which the area of the polygon is maximum. Formulate the optimization problem and use fundamental optimization principles systematically to find the optimal number of sides and corresponding side length of the polygon. Justify your result.

**4-19** Solve Problem 4-18 using the method of multiplier technique. Use five iterations of the MOM technique with  $R = 1$ . Compare the resulting multiplier  $\sigma_1$  with the Lagrange multiplier  $u_1$  obtained using the Kuhn-Tucker conditions.

**4-20** Consider the NLP problem:

$$\text{Maximize } f(x) = x_1 + x_2^2$$

subject to

$$25 - x_1^2 - x_2^2 \geq 0,$$

$$x_1^2 + x_2 \leq 9,$$

$$0 \leq x_1 \leq 5,$$

$$0 \leq x_2 \leq 10.$$

- (i) Set up and solve the first subproblem for the cutting plane method.
- (ii) Calculate the cutting plane at the resulting point.
- (iii) Set up and solve the second subproblem and generate the next cut.

In all cases, show the cutting planes on an  $x_1$ - $x_2$  plot.

**4-21** Explain why for the following NLP problem, the Lagrange multiplier  $u$  for the inequality constraint need to be nonnegative:

$$\text{Minimize } f(\mathbf{x}),$$

subject to

$$g(\mathbf{x}) \geq 0.$$

**4-22** Use two iterations of the cutting plane method to solve the following problem:

$$\text{Maximize } f(x) = x_2$$

subject to

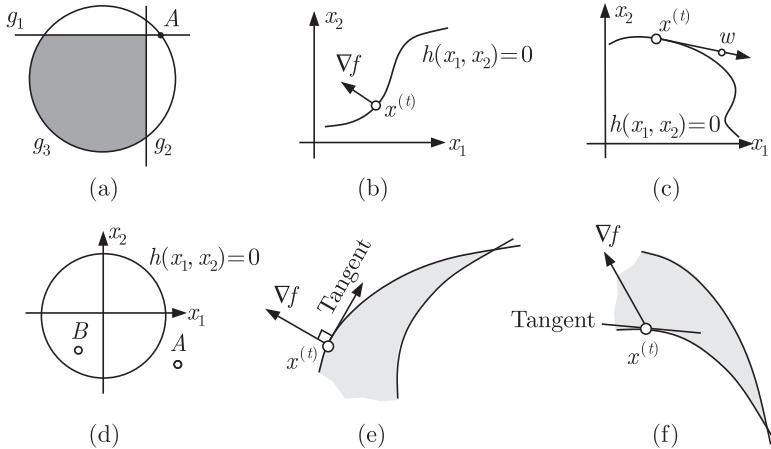
$$4.5x_1 + x_2^2 \leq 18,$$

$$2x_1 - x_2 \geq 1,$$

$$x_1, x_2 \geq 0.$$

Choose a suitable initial feasible region.

**4-23** Answer the following questions:



- (i) The feasible region bounded by two linear inequality constraints ( $g_1$  and  $g_2$ ) and one nonlinear inequality constraint (circle) is shown shaded. If point  $A$  is the current point, can you draw the cutting plane which will be formed from these three constraints?
- (ii) If  $x_1$  is considered as a nonbasic variable in a two-variable minimization problem having one equality constraint ( $h(\mathbf{x}) = 0$ ), indicate the reduced gradient direction in the figure at the point  $\mathbf{x}^{(t)}$  indicated. The gradient direction at this point is shown.
- (iii) Using the reduced gradient method, the point  $w$  is found from  $\mathbf{x}^{(t)}$  on the equality constraint  $h(\mathbf{x}) = 0$ . How would this point be made feasible if  $x_1$  is considered as the basic variable? Show the corresponding feasible point on the figure.
- (iv) The minimum distance of a point  $A$  from a given circle ( $x_1^2 + x_2^2 = a^2$ ) is needed to be found. If an equivalent minimization problem is formed with one equality constraint ( $h(x_1, x_2) = x_1^2 + x_2^2 - a^2 = 0$ ), what is the sign of the Lagrange multiplier of this constraint at the minimum solution? What would be the sign of the Lagrange multiplier if point  $B$  is used, instead of  $A$ ?
- (v) For the feasible direction method applied to a minimization problem, the gradient direction is shown at the current point. What will be the resulting search direction?
- (vi) In another minimization problem by the feasible direction method, the location of the current point is shown. Indicate the region on which the resulting search direction must lie.

**4-24** Find whether the search direction  $\hat{d}$  is feasible at point  $x$  on the constraint  $g(x)$ :

- (i)  $\hat{d} = (1, -2)^T$ ,  $x = (1, -0.5)^T$ ,  $g(x) = x_1^3 - 2x_1x_2 \geq 2$ .

- (ii)  $\hat{d} = (1, 2, 1)^T$ ,  $x = (0, 0, 0)^T$ ,  
 $g(x) = x_1^2 + x_2^2 - 3x_3 + 10x_1 \leq 0$ .
- (iii)  $\hat{d} = (0, 1)^T$ ,  $x = (2, 1)^T$ ,  
 $g(x) = (x_1 - 2x_2)^2 + 5x_1^2 x_2 + x_2 - 12 \leq 0$ .
- (iv)  $\hat{d} = (-1, -10)^T$ ,  $x = (1, 2)^T$ ,  
 $g(x) = 2 \exp(2x_1^2 - x_2) + x_2^2 \geq 6$ .
- (v)  $\hat{d} = (1, 0, -1)^T$ ,  $x = (1, 1, 1)^T$ ,  
 $g(x) = (x_1 + 2x_2 - x_3)^2 + (x_2 + 2x_3)^3 \leq 31$ .

**4-25** In an NLP problem, the following constraints are used:

$$\begin{aligned} g_1(x) &= (x_1 - 5)^2 + x_2^2 - 25 \leq 0, \\ g_2(x) &= x_1 + 2x_2 - 12 \leq 0, \\ g_3(x) &= 2x_1 + x_2 + 4 \geq 0. \end{aligned}$$

The current point is  $(0, 0)^T$  and a search direction  $\hat{d} = (2, 1)^T$  is found. Find the minimum and maximum allowable bounds along this direction.

**4-26** Consider the problem:

$$\text{Minimize } f(x) = (x - 5)^2$$

subject to

$$2 \leq x \leq 3.$$

Use Zoutendijk's feasible direction method to solve the above problem. Use a starting point  $x^{(0)} = 2$ . Show the proceedings of the algorithm on a  $f(x)$ - $x$  plot.

**4-27** For an optimization problem, the following constraint set is given:

$$g(x) = x_1^2 + 4x_2^2 - 4 \geq 0, \quad 0 \leq x_1 \leq 3.$$

The current point is  $x^{(t)} = (1, 0.5)^T$ .

- (i) Construct a cutting plane at  $x^{(t)}$ .
- (ii) Show that the cut will eliminate a portion of the feasible region by checking the feasibility of an original feasible point.

**4-28** Complete one iteration of the reduced gradient technique to find the point  $x^{(1)}$  in solving the following NLP problem:

$$\text{Minimize } x_1^2 + 2x_2^2$$

subject to

$$x_1^2 + x_2^2 = 5.$$

Use a starting feasible solution  $x^{(0)} = (1, -2)^T$ , initial  $\alpha = 0.25$ , and all termination factors equal to 0.01. Show the intermediate points on an  $x_1$ - $x_2$  plot.

**4-29** Repeat Problem (4.28) using the gradient projection method.

**4-30** We would like to solve the following problem:

$$\text{Minimize } 2x_1^2 + x_2^2 - 20x_1 - 12x_2,$$

subject to

$$x_1^2 + 4x_2 - 20 = 0.$$

We would like to use the generalized reduced gradient method (GRG) to solve the problem using  $x_2$  as the basic variable and using  $x^{(0)} = (2, 4)^T$ .

- (i) Find the normalized search direction  $\bar{s}^{(0)}$ .
- (ii) For a solution  $x(\alpha)$  ( $= x^{(0)} + \alpha\bar{s}^{(0)}$ , use  $\alpha = 2$ ) along this direction  $\bar{s}^{(0)}$ , find the corresponding feasible point.
- (iii) Show the proceedings of the above procedure by showing a neat sketch of the feasible region.

**4-31** Why does the optimal Lagrange multiplier corresponding to an equality constraint take either a positive or a negative sign?

**4-32** Solve the following problem using QP method:

$$\text{maximize } f(x) = x^2, \quad \text{subject to } 0 \leq x \leq 1.$$

**4-33** Consider the following optimization problem:

$$\text{Minimize } (x_1 - 1)^2 + (x_2 - 1)^2,$$

subject to

$$x_1 - x_2 - 2 = 0,$$

$$x_1 + x_2 - 0.5 \leq 0.$$

Find the optimal solution using QP algorithm. Show each intermediate LP solution on a  $x_1$ - $x_2$  plot.

**4-34** Solve the following NLP problem

$$\text{Minimize } 2x_1^2 + x_2^2 - 20x_1 - 12x_2,$$

subject to

$$x_1^2 + 4x_2 - 20 \leq 0,$$

using the sequential quadratic programming (SQP) method. Start from  $\mathbf{x}^{(0)} = (1, 1)^T$ .

**4-35** Solve the following problem using SQP method:

$$\text{Maximize } (x_1 - 1.5)^2 + (x_2 - 4)^2$$

subject to

$$4.5x_1 + x_2^2 \leq 18,$$

$$2x_1 - x_2 \geq 1,$$

$$x_1, x_2 \geq 0.$$

Start from  $\mathbf{x}^{(0)} = (0, 0)^T$ .

**4-36** Solve the following problem using SQP method by starting from  $x^{(0)} = \pi$ :

$$\text{Minimize } \sin(x)$$

subject to

$$\cos(x) \geq 0,$$

$$0 \leq x \leq 2\pi.$$

**4-37** Use the gradient projection method to find the optimal solution of the following NLP problem:

$$\text{Minimize } f(x, y) = x^2 + y^2 - 6x - 2y + 2$$

subject to

$$y \leq x,$$

$$x + 5y \leq 15,$$

$$y \geq 0.$$

Begin your search from  $x^{(0)} = (0, 0)^T$ . Use the active constraint strategy. Show each step of the algorithm clearly and show the intermediate solutions on an  $x$ - $y$  plot.

*Help:* Projection matrix is  $(I - A^T(AA^T)^{-1}A)$  and the Lagrange multiplier vector is  $(AA^T)^{-1}A\nabla f$ .

**4-38** In Problem (4-28), the equality constraint is changed into an inequality constraint as follows:

$$\text{Minimize } x_1^2 + 2x_2^2$$

subject to

$$g(x) = x_1^2 + x_2^2 \geq 5.$$

Execute one iteration of the reduced gradient method using

- (i) slack variable strategy,
- (ii) active constraint strategy.

Use the same parameter values as those in Problem (4-28).

**4-39** Consider the problem:

$$\text{Minimize } f(x) = (x_1 - 1)^2 + (x_2 - 1)^2 + (x_3 - 0.5)^2$$

subject to

$$h(x) = x_1^2 + x_2^2 + x_3^2 - 1 = 0.$$

Recognizing that both the objective function and constraint surface are spherical, calculate the optimum point using elementary calculus and analytical geometry. In order to use the generalized reduced gradient (GRG) method on this problem, we choose to use  $\bar{x} = (x_1, x_2)$  and  $\hat{x} = (x_3)$  as nonbasic and basic variables, respectively. For a feasible initial point  $x^{(0)}$ , the nonbasic variable is  $\bar{x}^{(0)} = (0.3, 0.3)$ .

- (i) Calculate the direction vector  $d$ .
- (ii) Assuming  $x^{(1)} = x^{(0)} + \alpha d$ , where  $d = (\bar{d}, \hat{d})$ , find  $x^{(1)}$  that exactly minimizes  $f(x^{(1)})$ .
- (iii) Is this solution feasible? Explain by showing points on a three-dimensional sketch of the constraint surface.
- (iv) In order to obtain a feasible point from  $x^{(1)}$ , the point is projected on the constraint surface. What is the resulting point? Compare this solution with the true optimum point.

**4-40** In trying to solve the following problem using the feasible direction method at the point  $x^{(5)} = (1, 1)^T$ , a direction vector  $d^{(5)} = (1, 1/7)^T$  is obtained.

$$\text{Maximize } (x_1 - 1.5)^2 + (x_2 - 4)^2$$

subject to

$$4.5x_1 + x_2^2 \leq 18,$$

$$2x_1 - x_2 \geq 1,$$

$$x_1, x_2 \geq 0.$$

Use two iterations of the golden section search method to bracket the minimum point along the direction  $d^{(5)}$ . Assume the mid-point of that interval as the new point  $x^{(6)}$ . Create a search direction at the new point  $x^{(6)}$ .

**4-41** Consider the following optimization problem:

$$\text{Minimize } (x_1 - 2)^2 + (x_2 - 2)^2,$$

subject to

$$3 - x_1 - x_2 \geq 0,$$

$$10x_1 - x_2 - 2 \geq 0.$$

Starting from  $x^{(0)} = (0.2, 0)^T$ , perform one iteration of the feasible direction method. Show that the search direction obtained by the feasible direction method is within the region bounded by the steepest descent and the most feasible directions. Solve the resulting LP problem graphically.

**4-42** Complete one iteration of the gradient projection method to find the point  $x^{(1)}$  for the following NLP problem:

$$\text{Minimize } (x_1 - 1)^2 + 4(x_2 - 3)^2$$

subject to

$$x_1^2 + x_2^2 = 5.$$

Use a starting feasible solution  $x^{(0)} = (1, 2)^T$ , an initial  $\alpha = 0.2$ , and a termination factor equal to 0.01. Show intermediate points on an  $x_1$ - $x_2$  plot.

**4-43** Repeat Problem 4-42 using the generalized reduced gradient method. Handle the constraint using the slack variable strategy. Use other parameters same as that in Problem (4-28).

## COMPUTER PROGRAM

A computer code implementing the penalty function method is presented here. Cauchy's steepest descent method is used for multivariable function optimization and a combination of the bounding phase and the golden section search is used to achieve a unidirectional search. Other algorithms discussed in this chapter can also be easily coded. In the following, we first outline the code and then present results from a simulation run. The objective function is coded in the function `funct`. In the version given here, the Himmelblau function is coded as the objective function. The constraint used in Exercise 4.1.1 is coded. For any other objective function and constraints, the function `funct` can be suitably modified.

```

c      ****
c      *
c      *          PENALTY FUNCTION METHOD
c      *
c      ****
c Developed by Kalyanmoy Deb
c                      Indian Institute of Technology, Kanpur
c All rights reserved. This listing is for personal use.
c%%%%%%%

```

```

c This routine finds the minimum point of a constrained
c optimization problem using penalty function method.
c Code your function in subroutine funct
C%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      implicit real*8 (a-h,o-z)
      dimension x0(10),xstar(10),gr0(10),s0(10),x1(10),
      -          xd(10)
      common/constr/nc,r
      data n,nc/2,1/
c.....n is the number of variables
c.....nc is the number of constraints
c.....step 1 of the algorithm
      write(*,*) 'enter accuracy in steepest descent'
      read(*,*) eps
      write(*,*) 'enter accuracy in golden section'
      read(*,*) epss
      write(*,*) 'enter number of sequences'
      read(*,*) maxseq
      write(*,*) 'enter c: R^(t+1) = c R^(t)'
      read(*,*) c
      write(*,*) 'enter initial vector'
      read(*,*) (x0(i),i=1,n)
      write(*,*) 'enter initial R'
      read(*,*) r
      write(*,*) 'enter 1 for intermediate results'
      read(*,*) ip
      nfun = 0
      iseq = 1

c.....step 2 of the algorithm
100  write(*,1) iseq
1    format(15x,'Sequence Number = ',i4)
      write(*,*) '=====',
      -           '=====',
      write(*,6) (x0(i),i=1,n)
6    format(2x,'Starting Solution is: ',4(1pe14.4,','))
      call funct(n,x0,fstar,nfun)
      write(*,7) fstar
7    format(2x,'Function value: ',1pe13.5)
c.....step 3 of the algorithm
      call steepest(n,eps,epss,x0,xstar,fstar,nfun,ierr,
      -             gr0,s0,x1,xd,ip)
      if (ierr .ne. 1) then
      write(*,*) '-----',
      -           '-----',
      write(*,2) (xstar(i),i=1,n)

```

```

2   format(2x,'The minimum point is: ',4(1pe14.4,','))
    write(*,3) fstar
3   format(2x,'Function value: ',1pe13.5)
    write(*,4) nfun
4   format(2x,'Total function evaluations so far:',i8)
    write(*,*) '=====',
    -           '=====',
    else
      write(*,*) 'Program is terminated due to error.'
      stop
    endif
c.....step 4 of algorithm (a diff. criterion is used)
  if (iseq .ge. maxseq) go to 200
c.....step 5 of algorithm
  r = c * r
  do 101 i = 1,n
    x0(i) = xstar(i)
101 continue
  iseq = iseq + 1
  go to 100
200 stop
end

  subroutine steepest(n,eps,epss,x0,xstar,fstar,
  -                   nfun,ierr,grad0,s0,x1,xdummy,iprint)
c.....steepest Descent method
c.....n : dimension of design vector
c.....eps : accuracy in steepest-descent method
c.....epss: accuracy in golden section search
c.....x0 : initial design vector
c.....xstar : final design solution (output)
c.....fstar : final objective function value (output)
c.....nfun : number of function evaluations required
c.....ierr : error code, 1 for error
c.....rest all are dummy variables of size n
  implicit real*8 (a-h,o-z)
  dimension x0(n),xstar(n),grad0(n),s0(n),x1(n),
  -           xdummy(n)
  maxiter = 10000
  k = 0
c.....step 2 of the algorithm
2  call fderiv(n,x0,grad0,f0,nfun,xdummy)
  if (iprint .eq. 1) then
    write(*,*) '-----',
    -           '-----',
    write(*,8) k,(x0(i),i=1,n)

```

```

8   format(2x,'Iteration: ',i5,/,5x,
-           'Solution vector: ',4(1pe12.4,','))
9   format(5x,'Function value : ',1pe13.4,
-           ' Function Eval. : ',i7)
  endif
c.....step 3 of the algorithm
  call unitvec(n,grad0,gradmag)
  if ((gradmag .le. eps) .or. (k .ge. maxiter)) then
    do 11 i = 1,n
11    xstar(i) = x0(i)
    fstar = f0
    return
  endif
c.....step 4 of the algorithm
  do 10 i = 1,n
    s0(i) = -1.0 * grad0(i)
10  continue
  call bphase(n,x0,s0,a,b,nfun,xdummy)
  call golden(n,x0,s0,a,b,epss,alfastr,nfun,ierr,
-             xdummy)
  sum = 0.0
  do 12 i = 1,n
    x1(i) = x0(i) + alfastr * s0(i)
    if (dabs(x0(i)) .gt. eps) then
      sum = sum + dabs(x1(i) - x0(i))/x0(i)
    else
      sum = sum + dabs(x1(i) - x0(i))/eps
    endif
12  continue
  call funct(n,x1,f1,nfun)
c.....step 5 of the algorithm
  if (sum .le. eps) then
    do 14 i = 1,n
      xstar(i) = x1(i)
14  continue
    fstar = f1
    return
  else
    k = k + 1
    do 15 i = 1,n
      x0(i) = x1(i)
15  continue
    go to 2
  endif
end

```

```

      subroutine golden(n,x,s,a,b,eps,xstar,nfun,ierr,
      -                   xdummy)
c.....golden section search algorithm
c.....finds xstar such that f(x + xstar * s) is minimum
c.....x : solution vector
c.....s : direction vector
c.....a,b : lower and upper limits
c.....nfun : function evaluations
c.....ierr : error code, 1 for error
c.....xdummy : dummy variable of size n
      implicit real*8 (a-h,o-z)
      real*8 lw,x(n),s(n),xdummy(n)
c.....step 1 of the golden section search
      xstar = a
      ierr=0
      maxfun = 10000
      aw=0.0
      bw=1.0
      lw=1.0
      k=1
c.....golden number
      gold=(sqrt(5.0)-1.0)/2.0
      w1prev = gold
      w2prev = 1.0-gold
c.....initial function evaluations
      call mapfun(n,x,s,a,b,w1prev,fw1,nfun,xdummy)
      call mapfun(n,x,s,a,b,w2prev,fw2,nfun,xdummy)
      ic=0
c.....Step 2 of the golden section search
10   w1 = w1prev
      w2 = w2prev
c.....calculate function value for new points only
      if (ic .eq. 1) then
          fw2 = fw1
          call mapfun(n,x,s,a,b,w1,fw1,nfun,xdummy)
      else if (ic .eq. 2) then
          fw1 = fw2
          call mapfun(n,x,s,a,b,w2,fw2,nfun,xdummy)
      else if (ic .eq. 3) then
          call mapfun(n,x,s,a,b,w1,fw1,nfun,xdummy)
          call mapfun(n,x,s,a,b,w2,fw2,nfun,xdummy)
      endif
c.....region-elimination rule
      if (fw1 .lt. fw2) then
          ic = 1
          aw = w2

```

```

lw = bw-aw
w1prev = aw + gold * lw
w2prev = w1
else if (fw2 .lt. fw1) then
  ic = 2
  bw = w1
  lw=bw-aw
  w1prev = w2
  w2prev = bw - gold * lw
else
  ic = 3
  aw = w2
  bw = w1
  lw = bw-aw
  w1prev = aw + gold * lw
  w2prev = bw - gold * lw
endif
k=k+1
c.....step 3 of the golden section search
if (dabs(lw) .lt. eps) then
  xstar = a + (b-a) * (aw+bw)/2
  return
else if (nfun .gt. maxfun) then
  write(*,3) maxfun, a+aw*(b-a), a+bw*(b-a)
3   format(' The algorithm did not converge in',i6,
        -           ' function evaluations',/,' Interval (',
        -           1pe12.5,',',1pe12.5,')')
  ierr = 1
  return
endif
go to 10
end

subroutine bphase(n,x,s,a,b,nfun,xdummy)
c.....bounding phase method
c.....all arguments are explained in subroutine golden
implicit real*8 (a-h,o-z)
dimension x(n),s(n),xdummy(n)
c.....step 1 of the algorithm
c.....initial guess, change if you like
w0 = 0.0
delta = 1.0
1 call mapfun(n,x,s,0d0,1d0,w0-delta,fn,nfun,xdummy)
call mapfun(n,x,s,0d0,1d0,w0,f0,nfun,xdummy)
call mapfun(n,x,s,0d0,1d0,w0+delta,fp,nfun,xdummy)

```

```

c.....step 2 of the algorithm
  if (fn .ge. f0) then
    if (f0 .ge. fp) then
      delta = 1 * delta
    else
      a = w0 - delta
      b = w0 + delta
    endif
  elseif ((fn .le. f0) .and. (f0 .le. fp)) then
    delta = -1 * delta
  else
    delta = delta / 2.0
    go to 1
  endif
  k=0
  wn = w0 - delta
c.....step 3 of the bounding phase algorithm
  3  w1 = w0 + (2**k) * delta
     call mapfun(n,x,s,0.0d0,1.0d0,w1,f1,nfun,xdummy)
c.....step 4 of the bounding phase algorithm
  if (f1 .lt. f0) then
c.....bracketing isn't done, reset wn to w0 and w0 to w1
    k = k+1
    wn = w0
    fn = f0
    w0 = w1
    f0 = f1
    go to 3
  else
c.....bracketing is complete, so quit
    a = wn
    b = w1
  endif
  if (b .lt. a) then
    temp = a
    a = b
    b = temp
  endif
  return
end

  subroutine fderiv(n,x,grad,f,nfun,xd)
c.....derivative calculation at point x
  implicit real*8 (a-h,o-z)
c.....calculates the first derivative of the function
  dimension x(n),grad(n),xd(n)

```

```

      do 10 i = 1,n
         xd(i) = x(i)
10   continue
         call funct(n,xd,f,nfun)
c.....set delta_x
      do 12 i = 1,n
         if (xd(i) .lt. 0.01) then
            dx = 0.01
         else
            dx = 0.01 * xd(i)
         endif
         xd(i) = xd(i) + dx
         call funct(n,xd,fp,nfun)
         xd(i) = xd(i) - 2 * dx
         call funct(n,xd,fn,nfun)
         grad(i) = (fp-fn)/(2.0*dx)
         xd(i)=x(i)
12   continue
         return
      end

      subroutine mapfun(n,x,s,a,b,w,f,nfun,xd)
c.....first, a unit is mapped (for golden section)
c.....then, a point is found in s direction
      implicit real*8 (a-h,o-z)
      dimension x(n),s(n),xd(n)
      xw = a + w * (b-a)
      do 10 i = 1,n
         xd(i) = x(i) + xw * s(i)
10   continue
      call funct(n,xd,f,nfun)
      return
      end

      subroutine funct(n,x,f,nfun)
c.....calculates the function value at x
      implicit real*8 (a-h,o-z)
      dimension x(n),g(10)
      common/constr/nc,r
      nfun = nfun + 1
c.....code your objective function and constraints here
c.....objective function value
      f=(x(1)*x(1)+x(2)-11.0)**2+(x(1)+x(2)*x(2)-7.0)**2
c.....constraints
      g(1) = -26.0 + (x(1)-5.0)**2 + x(2)*x(2)

```

```

c.....form penalty function (bracket operator)
do 1 i = 1,nc
    if (g(i) .lt. 0.0) f = f + r * g(i)*g(i)
1  continue
return
end

subroutine unitvec(n,x,sum)
c.....finds unit vector and magnitude of a vector s
implicit real*8 (a-h,o-z)
dimension x(n)
sum = 0.0
do 1 i = 1,n
    sum = sum + x(i)*x(i)
1  continue
sum = dsqrt(sum)
if (sum .ge. 1e-06) then
    do 2 i = 1,n
        x(i) = x(i)/sum
2  continue
endif
return
end

```

## Simulation Run

The above code is run on a PC-386 under Microsoft FORTRAN for minimizing the constrained Himmelblau function starting from  $x^{(0)} = (0, 0)^T$ . Intermediate solutions obtained for the first five sequences of the penalty function method are shown.

```

enter accuracy in steepest descent
0.001
enter accuracy in golden section
0.001
enter number of sequences
5
enter c: R^(t+1) = c R^(t)
10
enter initial vector
0 0
enter initial R
0.01
enter 1 for intermediate results
0

```

```
Sequence Number = 1
=====
Starting Solution is: 0.0000E+00, 0.0000E+00,
Function value: 1.70010E+02

The minimum point is: 2.9639E+00, 2.0610E+00,
Function value: 3.16851E+00
Total function evaluations so far: 134
=====

Sequence Number = 2
=====
Starting Solution is: 2.9639E+00, 2.0610E+00,
Function value: 3.10671E+01

The minimum point is: 2.6280E+00, 2.4750E+00,
Function value: 2.59956E+01
Total function evaluations so far: 240
=====

Sequence Number = 3
=====
Starting Solution is: 2.6280E+00, 2.4750E+00,
Function value: 2.08695E+02

The minimum point is: 1.0111E+00, 2.9391E+00,
Function value: 5.86645E+01
Total function evaluations so far: 425
=====

Sequence Number = 4
=====
Starting Solution is: 1.0111E+00, 2.9391E+00,
Function value: 7.76006E+01

The minimum point is: 8.5058E-01, 2.9421E+00,
Function value: 6.02360E+01
Total function evaluations so far: 556
=====

Sequence Number = 5
=====
Starting Solution is: 8.5058E-01, 2.9421E+00,
Function value: 6.16682E+01

The minimum point is: 8.4169E-01, 2.9485E+00,
Function value: 6.03703E+01
Total function evaluations so far: 609
=====
```

# 5

## Specialized Algorithms

---

There exist a number of specialized optimization algorithms developed to solve specific types of problems. The algorithms described in the previous chapter can also be used to solve such problems, but the computational effort required to obtain the desired accuracy may be enormous. In certain problems, the algorithms of Chapter 4 cannot be used to find the correct solution. For example, consider the case of integer programming problems where decision variables can take integer values only. The penalty function method or any algorithm of Chapter 4 can be tried, but they may not find an integer solution because the integer restrictions of variables is not included in the algorithm. Thus, the optimization algorithms described in Chapter 4 may not be efficient to solve such problems.

In this chapter, two specialized optimization algorithms suitable for solving specific types of NLP problems are discussed. First, two integer programming algorithms are discussed. One of the methods is similar to the penalty function method described in Chapter 4. The other method relaxes the integer restrictions of integer variables and forms an NLP problem. Thereafter, the NLP problem is solved. Based on noninteger solution of each integer variable, the NLP is branched into two different but complementary NLP problems. This method is known as the branch-and-bound method. These algorithms can be easily extended to solve discrete-variable problems, where the variables are allowed to take any set of discrete values. Many engineering design problems require that the design variable take discrete values. For example, in an optimal truss design problem, the cross-section of a member is used as a design variable and it is desired that the members be chosen from the standard rolled-sections available in the market. Then, the design variable can only take a value from a chosen set of discrete values. The geometric programming method to solve problems expressed in *posynomial* forms is presented next. A posynomial of  $N$  variables is the product of variables raised to the power of some real values. In a posynomial, only positive values of the variables are allowed. Since the objective function and

constraints of many engineering design problems appear in posynomials, the geometric programming method is included here.

## 5.1 Integer Programming

In many design optimization problems, some variables are allowed to take integer values, whereas the rest may take real values. For example, in an optimal gear design problem, the number of teeth in the gear could be a integer design variable, whereas the width of the gear could be a real-valued design variable. In general, an integer nonlinear programming (INLP) problem with  $I$  integer variables can be written as follows:

$$\left. \begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & \\ g_j(x) \geq 0, & j = 1, 2, \dots, J; \\ h_k(x) = 0, & k = 1, 2, \dots, K; \\ x_i^{(L)} \leq x_i \leq x_i^{(U)}, & i = 1, 2, \dots, N; \\ x_i \text{ integers}, & i = 1, 2, \dots, I. \end{array} \right\} \quad (5.1)$$

A simple strategy would be to first solve the above problem assuming that all variables are permitted to take any real value and then to obtain enumerative solutions by rounding off the integer variables to their nearest integers. For example, if the problem contains two integer variables and if the solution obtained by relaxing the integer restrictions is  $(1.234, 5.678)^T$ , then we assume that the optimum point is one of following four solutions:  $(1, 5)^T$ ,  $(1, 6)^T$ ,  $(2, 5)^T$ , and  $(2, 6)^T$ . There are at least two difficulties with this strategy. First of all, if the number of integer variables is large, a large number of solutions are required to be checked to decide for the optimum. If there are  $n$  integer variables, this number is  $2^n$ . Furthermore, for some nonlinear functions, one of these solutions may not be the true optimum. Thus, an algorithm more sophisticated than the enumerative technique described above is necessary to solve integer programming problems efficiently.

The algorithms discussed in the previous chapters may also have difficulty in solving the problem stated in Equation (5.1). For gradient-based methods, numerical gradient computations will be erroneous, because any increment  $\Delta x_i$  smaller than one in any integer variable will make the solution infeasible<sup>1</sup>. Direct search methods will also fail because the basic idea of search direction fails in the case of discrete variables. However, random search methods (even though not efficient) may be used. In this section, we discuss two different

---

<sup>1</sup>However, in practice, integer programming problems are often solved using gradient-based methods with  $\Delta x_i = 1$ .

ways to handle integer programming problems. It is worth mentioning here that even though both algorithms are designed to solve integer programming problems, in principle, they can be easily modified and applied to solve problems having discrete decision variables. In the next chapter, we shall describe a more elegant way of solving discrete optimization problems.

### 5.1.1 Penalty Function Method

In the penalty function method, in addition to penalizing a solution for infeasible points violating the given constraints, solutions are also penalized for noninteger values of integer variables. The INLP problem described in Equation (5.1) can be solved using recursive optimization of the following penalty functions:

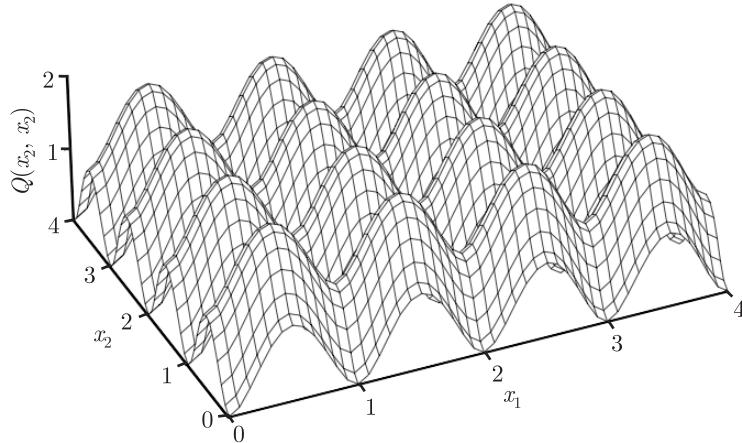
$$\begin{aligned} P(x, r_t, q_t) = & f(x) + r_t \left( \sum_{j=1}^J G_j(g_j(x)) + \sum_{k=1}^K H_k(h_k(x)) \right) \\ & + q_t \sum_{i=1}^I Q_t(x_i), \end{aligned} \quad (5.2)$$

where  $G_j(y)$  is a penalty term handling the  $j$ -th inequality constraint,  $H_k(y)$  is a penalty term handling the  $k$ -th equality constraint, and  $Q_t(x_i)$  is the penalty term handling the  $i$ -th integer variable. A number of penalty terms for handling inequality and equality constraints are discussed in Chapter 4 and can also be used here. But the penalty term  $Q_t$  is new and relates to the integer variables only. The penalty term  $Q_t$  for integer variables should be such that there is no penalty for the integer values but there is an increasing penalty for values away from integers. The following penalty term is suggested for this purpose (Rao, 1984):

$$Q_t(x_i) = [4(x_i - l_i)(1 - x_i + l_i)]^{\beta_t}, \quad (5.3)$$

where  $l_i = \lfloor x_i \rfloor$  (the operator  $\lfloor \cdot \rfloor$  takes the largest integer value smaller than the operand). Usually, a value of  $\beta_t \geq 1$  is used. In Figure 5.1, the above penalty function is shown for two integer variables. The figure shows that the penalty term  $Q_t$  is zero for integer values and increases as the value deviates from integer values. The penalty is maximum when the value is midway from integer values.

This method works similar to the penalty function method described in Chapter 4 except that in every sequence parameters  $r_t$ ,  $q_t$ , and  $\beta_t$  are all changed. The initial parameters are chosen so as to have a minimal distortion of the original function due to the addition of penalty terms in Equation (5.2). In successive sequences the parameter  $\beta_t$  is gradually reduced, the parameter  $q_t$  is gradually increased, and the parameter  $r_t$  is changed according to the penalty term used. Since the penalty function  $Q_t$  pushes the real values towards their nearest integers, a small tolerance parameter  $\epsilon_I$  is defined to check whether the current real value is close to an integer or not. If the real



**Figure 5.1** The penalty term  $Q_t$  for two integer variables.

solution is at most  $\epsilon_I$  away from an integer, the convergence to that integer solution is assumed. For example, if the current solution to an integer variable is 1.993 and the parameter  $\epsilon_I$  is set to be 0.01, then it is assumed that the integer variable takes the value 2, since the quantity  $|2 - 1.993|$  is smaller than  $\epsilon_I$ .

### Algorithm

**Step 1** Start with  $x^{(0)}$ ,  $r_1$ ,  $q_1$ , and  $\beta_1$ . Define a tolerance parameter  $\epsilon_I$  for integer variables. Set  $t = 1$ .

**Step 2** Formulate and minimize  $P(x, r_t, q_t)$  to compute  $x^{(t)}$ .

**Step 3** Is  $|P(x^{(t)}, r_t, q_t) - P(x^{(t-1)}, r_{t-1}, q_{t-1})|$  small? If not, update parameters  $r_{t+1}$ ,  $q_{t+1}$ , and  $\beta_{t+1}$ . Set  $t = t + 1$  and go to Step 2;

Else have all integer variables converged to integer values within the given tolerance? If yes, **Terminate**;

Else reset the values of  $r_t$  and  $q_t$  and go to Step 2.

We illustrate the working of this algorithm on a constrained integer nonlinear programming problem.

### EXERCISE 5.1.1

Consider the constrained INLP problem:

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 9)^2 + (x_1 + x_2^2 - 7)^2$$

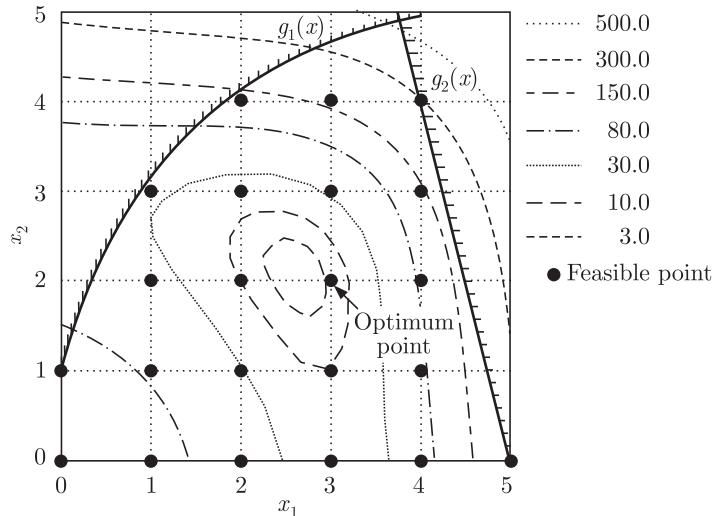
subject to

$$g_1(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

$$x_1, x_2 \geq 0, \quad x_1, x_2 \text{ integers.}$$

Notice that this objective function is different from the constrained Himmelblau function used in Chapter 4. If both variables can take real values, the minimum solution to the above problem is  $(2.628, 2.091)^T$  with a function value equal to zero. But for integer solution of  $x_1$  and  $x_2$ , the true minimum of the above INLP is  $(3, 2)^T$  with a function value equal to 4.0. Figure 5.2 shows the contour plot and the minimum point of the INLP problem.



**Figure 5.2** The feasible search space and the optimum point for the INLP problem in Exercise 5.1.1.

**Step 1** We begin with an initial solution  $x^{(0)} = (0, 0)^T$ . We choose

$$r_1 = 1.0, \quad \beta_1 = 2.0, \quad q_1 = 10.0.$$

We define the tolerance parameter  $\epsilon_I = 0.001$ . Thus, any solution which is  $\epsilon_I$  away from an integer is considered to take that integer value. We set the iteration counter  $t = 0$ .

**Step 2** The first penalty function is formulated as follows:

$$P(x, r_1, q_1) = f(x) + r_1 \sum_{i=1}^2 \langle g_j(x) \rangle^2 + q_1 \sum_{j=1}^2 Q_1(x_i).$$

Here, we are using the bracket-operator exterior penalty term. Recall that the quantity  $\langle \alpha \rangle$  takes a nonzero value  $\alpha$  only if  $\alpha < 0$ . The function  $Q_1$  is defined for  $\beta_1 = 2.0$ . Figure 5.3 shows the contour plot of the above function and the intermediate points found in optimizing the above penalized function using the steepest descent method. The initial point has a penalized function value equal to 130. The final point obtained is  $(2.928, 1.993)^T$  with a function equal to 2.463. This point is not feasible, as none of the variables have converged

to an integer value according to the tolerance parameter  $\epsilon_I$ . The penalized function value is found to be 3.185. This is a remarkable improvement from the initial solution. A few intermediate points of the steepest descent algorithm are shown in Table 5.1.

**Table 5.1** Intermediate Points Obtained Using the Steepest Descent Method for Successive Iterations of the Penalty Function Method

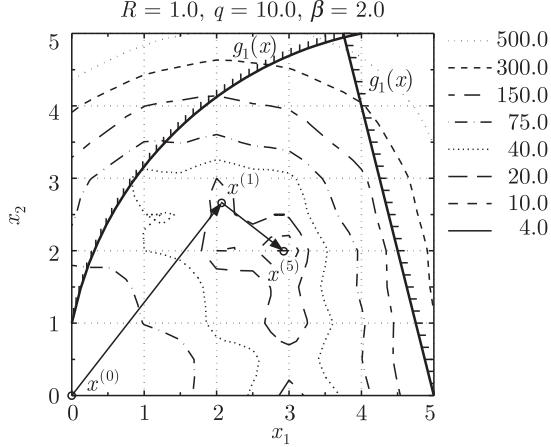
Sequence	$r_1$	$q_1$	$\beta_1$	Solution	Penalized function value
1	1.0	10.0	2.0	$(0, 0)^T$	130.000
				$(2.071, 2.662)^T$	17.561
				$(2.929, 1.993)^T$	3.185
				$\vdots$	$\vdots$
				$(2.928, 1.993)^T$	3.185
2	10.0	15.0	1.5	$(2.928, 1.993)^T$	4.605
				$(2.974, 2.006)^T$	3.968
				$\vdots$	$\vdots$
				$(2.975, 1.999)^T$	3.881
				$(2.925, 1.999)^T$	5.107
3	100.0	22.5	1.125	$(3.000, 1.999)^T$	4.039

**Step 3** Since this is the first sequence, we do not terminate. We reset the parameter values according to the following rules:  $r_2 = c_1 r_1$ ,  $q_2 = c_2 q_1$ ,  $\beta_2 = c_3 \beta_1$ , where  $c_1 = 10$ ,  $c_2 = 1.5$ , and  $c_3 = 0.75$  are chosen. Thus, the following parameter values are obtained:  $r_2 = 10.0$ ,  $q_2 = 15.0$ , and  $\beta_2 = 1.5$ . This completes one iteration of the penalty function method. We increment the iteration counter ( $t = 2$ ) and move to Step 2.

**Step 2** At this step, we formulate another penalty function:

$$P(x, r_2, q_2) = f(x) + r_2 \sum_{i=1}^2 \langle g_j(x) \rangle^2 + q_2 \sum_{i=1}^2 Q_2(x_i).$$

The function  $Q_2$  is defined for  $\beta_2$ . Starting from the solution found in the previous iteration  $(2.928, 1.993)^T$ , we use the steepest descent method and obtain the solution  $(2.975, 1.999)^T$  with a function value equal to 3.422. This point is also not feasible, but is closer to an integer solution than the initial point. The penalized function value at this point is 3.881.



**Figure 5.3** Intermediate points obtained using the steepest descent method on the penalized INLP problem.

**Step 3** Since the penalized value at this point is very different from that in the previous iteration, we do not terminate the algorithm. Instead, we update the parameters and move to Step 2 again. The updated parameters are  $r_3 = 100.0$ ,  $q_3 = 22.5$ , and  $\beta_3 = 1.125$ .

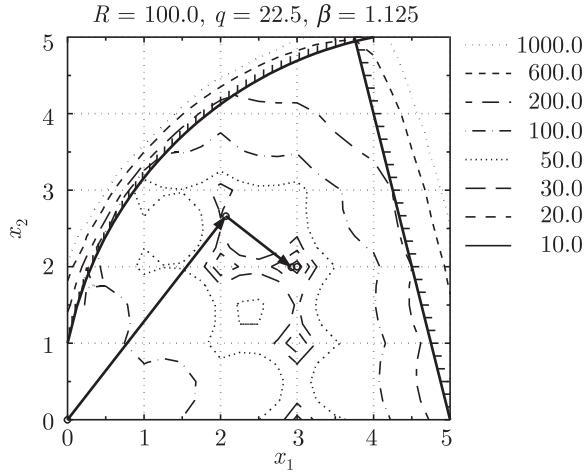
**Step 2** The new penalty function is formed with the above parameter values and solved from the starting point  $(2.975, 1.999)^T$ . The solution obtained at this stage is  $(3.000, 1.999)^T$  which can be approximated to be  $(3, 2)^T$  with the given tolerance  $\epsilon_I$ . Incidentally, this point is the true minimum of the original INLP problem.

**Step 4** Since the penalized values for two consecutive iterations are not close, the algorithm may proceed for one or two more iterations and finally terminate.

Figure 5.4 shows the distorted penalty function and the convergence of the solution to the true minimum. The above three iterations require a total of 374 function evaluations. This method, like all penalty function methods, suffers from the distortion of the original objective function—a matter which can be minimized by properly updating parameter values in successive iterations or by using the method of multiplier approach described in Chapter 4. The increase in values of the parameter  $q$  introduces multimodality in the function. But since in later sequences of the simulation the initial point is close to the true optimum, the penalty function method works successfully.

In discrete-variable problems where the design variables take discrete values in regular or irregular intervals, the penalty function  $Q_t(x_i)$  given in Equation (5.3) can be modified as follows:

$$Q_t(x_i) = \left[ 4 \left( \frac{x_i - l_i}{h_i - l_i} \right) \left( 1 - \frac{x_i - l_i}{h_i - l_i} \right) \right]^{\beta_t}, \quad (5.4)$$



**Figure 5.4** Intermediate points on a contour plot of the penalized function. Notice the multimodality of the function obtained due to the addition of the penalty function  $Q_3$ .

where  $l_i$  is the largest permissible discrete value smaller than  $x_i$  and  $h_i$  is the smallest permissible discrete value larger than  $x_i$ . For example, if a discrete variable takes values from the set  $[0.2, 0.4, 0.7, 1.0, 1.2, \dots]$ , and the current solution is  $x_i = 0.82$ , the above penalty function  $Q_t(x_i)$  is used with the following parameter values:  $l_i = 0.7$  and  $h_i = 1.0$ . It is interesting to note that for integer variables, the above equation reduces to Equation (5.3).

### 5.1.2 Branch-and-bound Method

In this method, the INLP problem is first solved by assuming that all variables are real-valued. Depending on the solution obtained, two different NLPs are formed for each integer variable with an extra constraint. Each of these two NLPs is further branched into two more NLPs. Depending on the outcome of an NLP solution, it is terminated or *fathomed*. When all branched NLPs are fathomed, the solution of the original INLP is found by checking the best of all NLP solutions. The algorithm is described below.

#### Algorithm

**Step 1** Assume that all variables are real. Solve the NLP problem. Let the solution be  $\{x_I^{(t)}, x_R^{(t)}\}$ . Initialize a counter  $k = 2$ .

**Step 2** For each unfathomed NLP, choose a particular integer variable  $i \in I$  for which  $x_i^{(t)}$  is a noninteger. Branch into two NLP problems each with an extra constraint. Form the first NLP (call it NLP- $k$ ) by adding the constraint  $x_i \leq \lfloor x_i^{(t)} \rfloor$  and form the second NLP (call it NLP-( $k + 1$ )) by adding the constraint<sup>2</sup>  $x_i \geq \lceil x_i^{(t)} \rceil$ . Solve NLP- $k$  and NLP-( $k + 1$ ). Increment  $k$  by 2.

<sup>2</sup>The operator  $\lceil \rceil$  takes the smallest integer value larger than the operand.

**Step 3** Fathom all NLPs one at a time, if one or more of the following conditions are satisfied:

- (i) The optimal solution of the NLP for all  $i \in I$  is integer valued.
- (ii) The NLP problem is infeasible.
- (iii) The optimal function value of the NLP is not better than the current best function value.

**Step 4** If all nodes have been fathomed, **Terminate**;

Else go to Step 2.

The selection of the integer variable to be branched is usually guided by one of the following rules (Reklaitis et al., 1983):

- (i) The integer variable corresponding to the better function value is used.
- (ii) The most important integer variable is used first (guided by the experience of the user).
- (iii) An arbitrary integer design variable is used.

When a number of NLP problems in any level are required to be solved, there is usually no preference for solving one NLP over the other. This is because all created NLPs in any level must be solved before proceeding to the next level.

We present hand simulation of this method on a numerical problem.

### EXERCISE 5.1.2

We consider the INLP problem used in the previous exercise problem and apply the branch-and-bound algorithm to solve the problem:

$$\text{Minimize } f(x) = (x_1^2 + x_2 - 9)^2 + (x_1 + x_2^2 - 7)^2$$

subject to

$$g_1(x) = 26 - (x_1 - 5)^2 + x_2^2 \geq 0,$$

$$g_2(x) = 20 - 4x_1 - x_2 \geq 0,$$

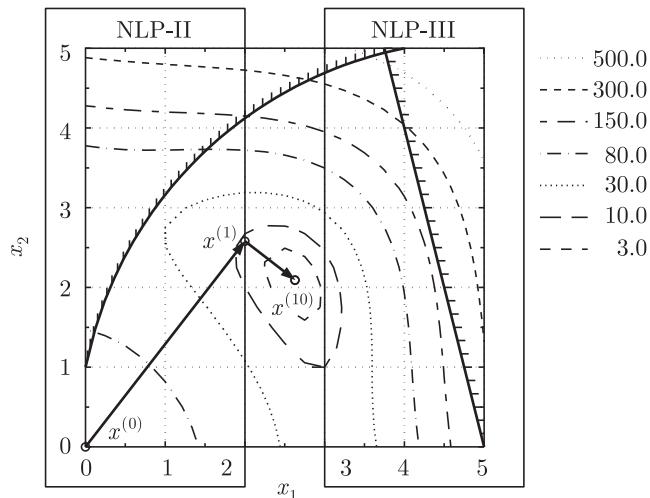
$$x_1, x_2 \geq 0, \quad x_1, x_2 \text{ integers.}$$

Recall that the optimal integer solution lies at the point  $(3, 2)^T$  with a function value equal to 4.0.

**Step 1** At first, we assume that both variables can take any real values. We solve the resulting NLP problem using the penalty function method described in the previous chapter starting from the initial point  $x^{(1)} = (0, 0)^T$ . We choose  $R^{(0)} = 1.0$  and obtain the solution  $x^{(1)} = (2.628, 2.091)^T$ . The second sequence is performed with  $R^{(1)} = 10.0$  and with  $x^{(1)}$  as the initial point. We obtain the point  $x^{(2)} = (2.628, 2.091)^T$  having an objective function value equal to  $f(x^{(2)}) = 4.152 \times 10^{-7}$ . Since these solutions  $x^{(1)}$  and  $x^{(2)}$  are the same up to three decimal places of accuracy, we terminate the search process. Intermediate points are tabulated in Table 5.2 and the points are shown on a contour plot in Figure 5.5. This solution requires 306 function evaluations.

**Table 5.2** Intermediate Points Obtained using the Penalty Function Method for NLP-I. (Steepest descent method is used as the unconstrained optimization technique.)

Sequence	Penalty parameter	Solution	Penalized function value
1	1.0	$(0, 0)^T$	130.000
		$(2.000, 2.580)^T$	8.494
		$(2.629, 2.095)^T$	$4.08(10^{-4})$
		$\vdots$	$\vdots$
		$(2.628, 2.091)^T$	$4.63(10^{-5})$
2	10.0	$(2.628, 2.091)^T$	$4.15(10^{-8})$



**Figure 5.5** Intermediate points for NLP-I on a contour plot of the objective function  $f(x)$  for real values of  $x_1$  and  $x_2$ . Based on the obtained optimum of NLP-I, the search space is divided into two nonoverlapping regions (NLP-II and NLP-III). Note that no feasible point in NLP-I is eliminated in forming the two NLPs. The choice of division along  $x_1$  is arbitrary.

**Step 2** At this point, both solutions are real-valued. Thus, we branch along any one of the variables. Let us assume that we choose to branch on  $x_1$  variable. Thus, we form two NLP problems as shown in Table 5.3. Let us solve the NLP-II problem first. We use the penalty function method with the steepest descent method as the unconstrained optimizer and the golden section

**Table 5.3** NLP-II and NLP-III Problems Formed from the Solution of NLP-I. (One extra constraint is added to the original INLP problem to form two NLP problems. Note that no feasible solution is eliminated.)

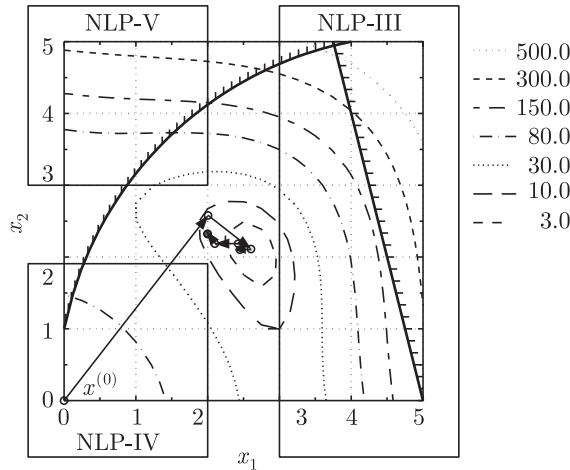
NLP-II	NLP-III
Minimize $f(x)$	Minimize $f(x)$
subject to	subject to
$g_1(x) \geq 0,$	$g_1(x) \geq 0,$
$g_2(x) \geq 0,$	$g_2(x) \geq 0,$
$x_1 \leq 2.0,$	$x_1 \geq 3.0,$
$x_1, x_2 \geq 0.$	$x_1, x_2 \geq 0.$

**Table 5.4** Intermediate Points Obtained using the Penalty Function Method for NLP-II

Sequence	Penalty parameter	Solution	Penalized function value
1	1.0	$(0, 0)^T$	130.000
		$(2.006, 2.580)^T$	8.494
		$(2.605, 2.113)^T$	0.381
		⋮	⋮
		$(2.603, 2.104)^T$	0.379
2	10.0	$(2.603, 2.104)^T$	3.650
		$(2.450, 2.104)^T$	2.839
		⋮	⋮
		$(2.427, 2.187)^T$	2.718
		$(2.427, 2.187)^T$	19.128
3	100.0	$(2.102, 2.187)^T$	6.787
		⋮	⋮
		$(2.091, 2.324)^T$	6.377
		$(2.001, 2.324)^T$	7.311
4	1000.0		

search as the unidirectional optimizer. We begin with an initial solution  $x^{(0)} = (0, 0)^T$  and  $R^{(0)} = 1.0$ . As shown in Table 5.4, four successive sequences

of the penalty function method find the minimum point  $(2.001, 2.324)^T$ . Since the solution to the first variable is within the tolerance level (which is assumed to be  $\epsilon_I = 0.001$ ), we set the solution to be  $(2, 2.324)^T$ . At this point, the function value is 7.322. The penalty function method requires 850 function evaluations. The intermediate points of the penalty function method are shown in Figure 5.6.



**Figure 5.6** Intermediate points for NLP-II and NLP-III on a contour plot of the objective function  $f(x)$  for real values of  $x_1$  and  $x_2$ . NLP-II is branched into NLP-IV and NLP-V based on the obtained solution of NLP-II.

After the solution of NLP-II, we solve the NLP-III problem from the same starting point and penalty parameters. This time, we obtain the solution  $(2.999, 1.925)^T$  (refer to Table 5.5). This solution can be accepted to be the solution  $(3, 1.925)^T$  with the specified tolerance level. At this point, the function value is 3.792. The number of function evaluations required to solve this problem is 952. Figure 5.6 shows how the penalty function method finds this minimum point.

**Step 3** Since none of the problems NLP-II and NLP-III satisfy any of the three conditions for fathoming a node, we proceed to Step 4.

**Step 4** Since NLP-II and NLP-III are not yet fathomed, we move to Step 2. This completes one iteration of the branch-and-bound method.

**Step 2** From each node (NLP-II and NLP-III), we now have to branch into two more nodes. This makes a total of four NLP problems. Recalling that the solution obtained from NLP-II is  $(2, 2.324)^T$ , we branch only on the second variable. Two problems arising from NLP-II are tabulated in Table 5.6. Using the penalty function method and starting from the point  $(0, 0)^T$ , we solve NLP-IV and obtain the solution  $(2.001, 2.000)^T$ . We approximate this solution to be  $(2, 2)^T$  with a function value equal to 10.0. This procedure requires 510 function evaluations.

**Table 5.5** Intermediate Points Obtained using the Penalty Function Method for NLP-III. (The steepest descent method is used as the unconstrained optimization technique.)

Sequence number	Penalty parameter	Solution	Penalized function value
1	1.0	$(0, 0)^T$	139.000
		$(2.510, 2.559)^T$	0.810
		$(2.653, 2.100)^T$	0.143
		$\vdots$	$\vdots$
		$(2.643, 2.083)^T$	0.132
2	10.0	$(2.643, 2.083)^T$	1.279
		$(2.723, 2.083)^T$	1.019
		$\vdots$	$\vdots$
		$(2.734, 2.035)^T$	0.983
3	100.0	$(2.734, 2.035)^T$	7.351
		$(2.911, 2.035)^T$	3.071
		$\vdots$	$\vdots$
		$(2.919, 1.925)^T$	2.887
4	1000.0	$(2.919, 1.925)^T$	8.791
		$(2.989, 1.925)^T$	3.670
		$\vdots$	$\vdots$
		$(2.999, 1.925)^T$	3.791

Solving NLP-V, we obtain the solution  $(1.999, 3.000)^T$ . This solution can be approximated as  $(2, 3)^T$  with a function value equal to 20.0. The number of function evaluations required to solve NLP-V is 612.

In NLP-III, the solution obtained is  $(3, 1.925)^T$ . The progress of this simulation is not shown in Figure 5.6, for the sake of brevity. We branch on the second variable only. Two NLP problems arising from NLP-III are shown in Table 5.7.

Solutions to these two problems are also found using the penalty function method from an initial point  $(0, 0)^T$ . The solution for NLP-VI is  $(3.000, 1.000)^T$  with a function value equal to 10.0 and the solution for NLP-VII is found to be  $(3.000, 2.001)^T$ . This solution is approximated to be  $(3, 2)^T$  with a function value equal to 4.0. The function evaluations required to solve these NLPs are 544 and 754, respectively.

**Table 5.6** NLP-IV and NLP-V Problems Formed from the Solution of NLP-II. (One extra constraint is added to NLP-II to form two NLP problems.)

NLP-IV	NLP-V
Minimize $f(x)$	Minimize $f(x)$
subject to	subject to
$g_1(x) \geq 0,$	$g_1(x) \geq 0,$
$g_2(x) \geq 0,$	$g_2(x) \geq 0,$
$x_1 \leq 2.0,$	$x_1 \leq 2.0,$
$x_2 \leq 2.0,$	$x_2 \geq 3.0,$
$x_1, x_2 \geq 0.$	$x_1, x_2 \geq 0.$

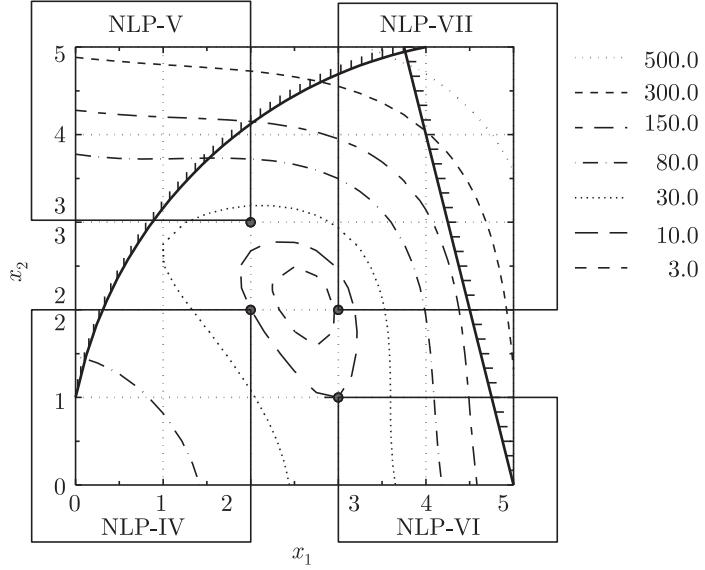
**Table 5.7** NLP-VI and NLP-VII Problems Formed from the Solution of NLP-III. (One extra constraint is added to NLP-III to form two NLP problems.)

NLP-VI	NLP-VII
Minimize $f(x)$	Minimize $f(x)$
subject to	subject to
$g_1(x) \geq 0,$	$g_1(x) \geq 0,$
$g_2(x) \geq 0,$	$g_2(x) \geq 0,$
$x_1 \geq 3.0,$	$x_1 \geq 3.0,$
$x_2 \leq 1.0,$	$x_2 \geq 2.0,$
$x_1, x_2 \geq 0.$	$x_1, x_2 \geq 0.$

**Step 3** Since the solution to NLP-IV is a feasible solution (having integer values for both variables), we fathom this node. Similarly, we fathom the node at NLP-V. The current best solution for problems NLP-IV and NLP-V is  $(2, 2)^T$  with a function value equal to 10.0. The intermediate points of the simulation are shown in Figure 5.7.

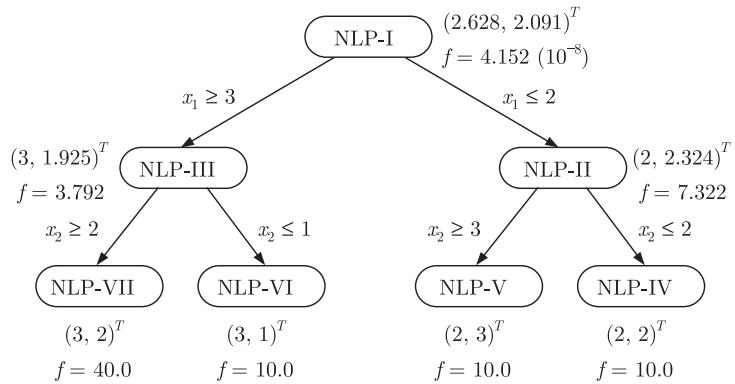
The solutions to both NLP-VI and NLP-VII are also feasible (integer valued). Thus, we fathom each of these NLPs. The current best solution at this stage is found as  $(3, 2)^T$  with a function value equal to 4.0. Two NLPs and their solutions using the penalty function method are shown in Figure 5.7. We now move to Step 4.

**Step 4** Since all open nodes are now fathomed, we terminate the algorithm. Therefore, the solution to the original INLP problem is  $(3, 2)^T$  with optimal function value equal to 4.0.



**Figure 5.7** The optimum points for NLP-IV, NLP-V, NLP-VI, and NLP-VII on a contour plot of the objective function  $f(x)$  for real values of  $x_1$  and  $x_2$ . No feasible solution in NLP-IV is eliminated.

The flowchart of the progress of the algorithm is outlined in Figure 5.8. The figure shows how an NLP is branched into two NLPs with an extra constraint and how an NLP is fathomed. The complete solution of the INLP problem requires a total of 4,528 function evaluations; whereas there are only 22 feasible points in the specified search space (Figure 5.2). This computational complexity of this algorithm increases as the number of integer variables increases. Due to this reason, it is advisable not to use this algorithm in problems where the number of integer variables exceeds ten or so.



**Figure 5.8** Different NLPs in the branch-and-bound method.

## 5.2 Geometric Programming

The geometric programming (GP) method was developed by C. Zener in 1961. GP methodology is designed to solve NLP problems which can only be expressed in *posynomial* form:

$$f(x) = \sum_{t=1}^T c_t \prod_{i=1}^N x_i^{a_{it}},$$

where  $c_t > 0$  and  $a_{it}$  is any real number (positive, negative, or zero). The terms are called posynomials because all variables  $x_i$  can take only positive values. We shall discuss more about functions with negative  $x_i$  values later. In the above expression,  $T$  posynomial terms are added together. Each of the posynomial terms may contain at most  $N$  design variables. Fortunately, many engineering design problems can be expressed in the above form and in those problems GP methodology works more efficiently than other methods (Beightler and Phillips, 1976).

In the GP method, the original optimization problem (supposedly in posynomial form) is known as the *primal* problem. If the original problem is not in posynomial form, variable substitutions may be used to convert the original problem into the primal form. Thereafter, the *primal* problem is converted to an equivalent *dual* problem expressed in terms of a set of dual variables. The transformation is achieved using the arithmetic-geometric-mean inequality which states that the arithmetic mean is greater than or equal to the geometric mean of a set of positive numbers. Using this inequality, it can be shown that for a set of variables  $\delta_t > 0$  (where  $t = 1, 2, \dots, T$ ), the following inequality is true (Rao, 1984):

$$f(x) \geq \prod_{t=1}^T \left( \frac{c_t}{\delta_t} \right)^{\delta_t} \quad (5.5)$$

with the assumption that

$$\sum_{t=1}^T \delta_t = 1,$$

$$\sum_{t=1}^T a_{it} \delta_t = 0 \quad \text{for all } i = 1, 2, \dots, N.$$

The equality sign in inequality (5.5) holds good when all  $\delta_t$  are equal (with a value  $1/T$ ). Thus, an  $N$ -dimensional, unconstrained primal problem can be substituted by an equivalent  $T$ -dimensional, linearly constrained dual problem. Consider the primal problem given below:

$$\text{Minimize } f(x) = \sum_{t=1}^T c_t \prod_{i=1}^N x_i^{a_{it}}$$

subject to

$$x_i > 0 \quad \text{for } i = 1, 2, \dots, N.$$

The posynomial objective function can be expressed in the dual form by using the arithmetic-geometric-mean inequality principle. It can be shown that this problem is equivalent to solving the following dual problem (Reklaitis et al., 1983):

$$\text{Maximize } Z(\delta) = \prod_{t=1}^T \left( \frac{c_t}{\bar{\delta}_t} \right)^{\delta_t}$$

subject to

$$\begin{aligned} \sum_{t=1}^T \delta_t &= 1, \\ \sum_{t=1}^T a_{it} \delta_t &= 0, \quad i = 1, 2, \dots, N, \\ \delta_t &\geq 0. \end{aligned}$$

Once the optimal dual variables ( $\delta^*$ ) are found by solving the above dual NLP problem, the corresponding primal solutions ( $x^*$ ) can be obtained by solving following linear simultaneous equations:

$$\sum_{i=1}^N a_{it} \ln [x_i^*] = \ln \left[ \frac{\delta_t^* Z(\delta^*)}{c_t} \right], \quad t = 1, 2, \dots, T. \quad (5.6)$$

In the above formulation, there are  $N$  unknown variables and  $T$  equations. The degree of difficulty of the algorithm varies depending on the value of  $T$  and  $N$ . The degree of difficulty in GP terminology is defined as  $d = (T - N - 1)$ .

If  $d < 0$ , the number of equality constraints is more than the number of dual variables. Therefore, a feasible solution may not exist. If a solution exists, *any*  $T$  equality constraints can be used to find the optimal solution  $\delta^*$ . The primal to dual formulation is such that for the optimal solution this always happens.

On the other hand, if  $d > 0$ , the number of equality constraints is less than the number of dual variables. Thus, some of the dual variables can be eliminated using the equality constraints. An optimization method (preferably the reduced gradient method or the gradient projection method discussed in the previous chapter) may be used to find the optimal dual variables.

However, if  $d = 0$ , there are exactly that many dual variables as the number of equality constraints. Those equations ( $T$  simultaneous equations) can be solved using the Gauss-elimination method (Strang, 1980) to find  $T$  dual variables.

An interesting relationship between primal and dual problems is that the optimal primal function value is the same as the optimal dual function value, or  $f(x^*) = Z(\delta^*)$ . The above GP method can also be extended to solve constrained optimization problems with constraints having posynomial terms. Avoiding the details, we first outline the primal problem and then present the corresponding dual problem.

### Primal problem

The primal problem must be in the following form:

$$\text{Minimize } f_0(x) = \sum_{t=1}^{T_0} c_{0t} \prod_{i=1}^N x_i^{a_{0it}}$$

subject to

$$f_j(x) = \sum_{t=1}^{T_j} c_{jt} \prod_{i=1}^N x_i^{a_{jit}} \leq 1, \quad j = 1, 2, \dots, J;$$

$$x_i > 0, \quad i = 1, 2, \dots, N.$$

Note that the constraints are also in posynomial form. The inequality constraints are required to be less-than-or-equal type and the right side of each inequality constraint must be equal to one. If the NLP problem has a greater-than-or-equal type constraint, it must be converted to a less-than-or-equal type constraint. Fortunately, in engineering problems, the inequality constraints occur primarily due to some resource limitation. Thus, the inequality constraints are mainly less-than-or-equal type and are suitable for GP methodology. Equality constraints are usually avoided from consideration. They can be included by relaxing the equality requirement and by using two inequality constraints, as discussed in Chapter 1.

The greatest difficulty in using GP methodology for constrained optimization problems lies in the formulation of constraints in the above form. However, if the objective function and constraints can be written in the above form, the optimization procedure is efficient. The number of dual variables in the above problem is the total number of terms in the objective function and constraints, or

$$T = \sum_{j=0}^J T_j.$$

In order to derive the dual problem, Lagrange multipliers ( $u_j$ ) for all inequality constraints are used. As we have discussed before, the Lagrange multiplier for an inequality constraint is the sum of the dual variables corresponding to each posynomial term in the left-side expression of the constraint. Thus, for the  $j$ -th inequality constraint having  $T_j$  terms, the Lagrange multiplier is

$$u_j = \sum_{t=1}^{T_j} \delta_{jt}.$$

The contribution of dual variables to the Lagrange multiplier of the constraint is proportional to the corresponding posynomial value. Thus, we

may write the dual variables for the objective function and constraints as follows:

$$\delta_{0t} = c_{0t} \prod_{i=1}^N x_n^{a_{0it}} / f_0(x),$$

$$\delta_{jt} = u_j c_{jt} \prod_{i=1}^N x_i^{a_{j it}}.$$

Using the dual variables and Lagrange multipliers, we now present the corresponding dual problem. The analysis for conversion of the primal problem to the dual problem is beyond the scope of this book. Interested readers may refer to a more advanced book (Rao, 1984).

### Dual problem

We write the dual problem in terms of the dual variables  $\delta$ . The original minimization problem becomes a maximization problem in terms of the dual variables. The resulting dual problem contains a number of linear constraints. One advantage of working with the dual problem is that the constraints are linear in terms of the dual variables. Thus, the Frank-Wolfe method discussed in Chapter 4 becomes suitable to solve the dual problem.

$$\text{Maximize } Z(\delta) = \prod_{j=0}^J \prod_{t=1}^{T_j} \left( \frac{c_{jt} u_j}{\delta_{jt}} \right)^{\delta_{jt}}$$

subject to

$$\sum_{t=1}^{T_0} \delta_{0t} = 1,$$

$$\sum_{j=0}^J \sum_{t=1}^{T_j} a_{j it} \delta_{jt} = 0, \quad i = 1, 2, \dots, N;$$

$$\delta_{jt} \geq 0, \quad j = 0, 1, 2, \dots, J, \quad t = 1, 2, \dots, T_j.$$

Here the Lagrange multiplier for the  $j$ -th inequality constraint is

$$u_j = \sum_{t=1}^{T_j} \delta_{jt}.$$

The first constraint in the above problem suggests that  $u_0 = 1$ . As before, once the optimal dual variables  $\delta_{jt}^*$  (and hence optimal Lagrange multipliers  $u_j^*$ ) are found, the corresponding primal variables can be found by solving the

following simultaneous linear equations:

$$\sum_{i=1}^N a_{0it} \ln(x_i^*) = \ln \left( \frac{\delta_{0t}^* f_0(x^*)}{c_{0t}} \right), \quad (5.7)$$

$$\sum_{i=1}^N a_{j�} \ln(x_i^*) = \ln \left( \frac{\delta_{jt}^*}{u_j^* c_{jt}} \right), \quad j = 1, 2, \dots, J. \quad (5.8)$$

Both the above equations are valid for  $t = 1, 2, \dots, T_m$  and  $\delta_{mt}^* > 0$ . The degree of difficulty in this problem is equal to

$$d = \sum_{j=0}^J T_j - N - 1.$$

It can be shown that all optimal dual variables corresponding to an inactive constraint are zero. With this background, we first present the GP algorithm and then show a hand-calculation for solving an NLP problem using the GP method.

### Algorithm

**Step 1** From the given primal problem, form the dual problem using the above procedure. Calculate the degree of difficulty.

**Step 2** Solve the dual problem to find the optimal dual variables  $\delta_{jt}^*$ . Depending on the degree of difficulty, an optimization method may be necessary.

**Step 3** At the optimal point,  $f(x^*) = Z(\delta^*)$ . The optimal primal variables can be calculated by solving Equations (5.7) and (5.8) at the optimal dual solution. **Terminate**.

In the GP methodology, the design variables  $x_i$  are required to be nonnegative. If any negative values of design variables are anticipated, a suitable transformation of the variable may be used to ensure that the new variable is always nonnegative. Usually, an exponential transformation  $z_i = \exp(x_i)$  is used for this purpose. The primal problem is written in terms of variable  $z_i$  and an equivalent dual problem is formed and solved. After the optimal primal variables are calculated, the original design variables are found by using the inverse transformation function:  $x_i = \ln(z_i)$ .

In the following, we present hand simulation of the constrained GP methodology to a numerical NLP problem with one constraint.

**EXERCISE 5.2.1**

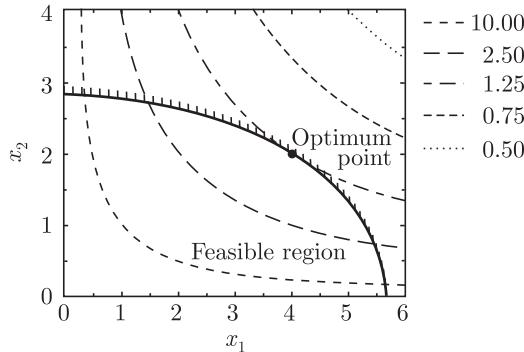
We consider the following two-variable NLP problem where both variables are nonnegative:

$$\text{Minimize } 10/(x_1 x_2)$$

subject to

$$x_1^2 + 4x_2^2 \leq 32, \quad x_1, x_2 > 0.$$

The contour of the objective function, the feasible space, and the minimum point are shown in Figure 5.9. The minimum point is  $(4, 2)^T$  with a function value equal to 1.25.



**Figure 5.9** The feasible search space and the optimum point.

**Step 1** In order to form the dual problem, we first find the degree of difficulty. We observe that there is only one constraint ( $J = 1$ ). We also observe that  $N = 2$ ,  $T_0 = 1$ , and  $T_1 = 2$ . Thus, the degree of difficulty is  $d = (1 + 2) - 2 - 1 = 0$ , which suggests that an optimization method is not required to solve the dual problem; the solution of simultaneous linear equations is sufficient to find the optimal solution.

We first write the primal problem in posynomial form and then present the corresponding dual problem.

**Primal problem**

$$\text{Minimize } f_0(x) = 10x_1^{-1}x_2^{-1}$$

subject to

$$f_1(x) = \frac{1}{32}x_1^2 + \frac{1}{8}x_2^2 \leq 1, \\ x_1, x_2 > 0.$$

**Dual problem**

$$\text{Maximize } Z(\delta) = 10^{\delta_{01}} \left( \frac{\delta_{11} + \delta_{12}}{32\delta_{11}} \right)^{\delta_{11}} \left( \frac{\delta_{11} + \delta_{12}}{8\delta_{12}} \right)^{\delta_{12}}$$

subject to

$$\begin{aligned}\delta_{10} &= 1, \\ (-1)\delta_{01} + (2)\delta_{11} &= 0, \\ (-1)\delta_{02} + (2)\delta_{12} &= 0, \\ \delta_{01}, \delta_{11}, \delta_{12} &\geq 0.\end{aligned}$$

**Step 2** From the equality constraints, we obtain the optimal values:  $\delta_{01}^* = 1$  and  $\delta_{11}^* = \delta_{12}^* = 0.5$ . In any other problem, the Gauss-elimination method may be used to find the optimal dual variables.

**Step 3** The optimal dual function value is  $Z(\delta^*) = 1.25$ , which is also equal to the optimal primal function value  $f_0(x^*)$ . In order to find the optimal primal variables, we write Equations (5.7) and (5.8) at the optimum point:

$$\begin{aligned}(-1) \ln x_1 + (-1) \ln x_2 &= \ln \left[ \frac{(1)(1.25)}{10} \right], \\ (2) \ln x_1 &= \ln \left[ \frac{0.5}{(1)(1/32)} \right], \\ (2) \ln x_2 &= \ln \left[ \frac{0.5}{(1)(1/8)} \right].\end{aligned}$$

From the last two equations, we obtain  $x_1^* = 4$  and  $x_2^* = 2$ . It is interesting to note that these values satisfy the first equation. We emphasize here that this did not happen accidentally. The formulation of the primal to dual problem is such that optimal primal variables can be calculated by using any  $N$  equations. The rest of the equations will be automatically satisfied. Thus, the minimum solution to the given problem is  $x^* = (4, 2)^T$  with a function value equal to 1.25.

The above formulation can solve problems where the objective function and constraints can be expressed by the summation of a number of posynomials. This restricts the application of the above method to a narrow class of problems. In order to apply the GP method to problems formed by addition and/or subtraction of posynomials, an extension of the original GP algorithm is suggested.

Let us consider the following primal problem which can be written by algebraic sum of several posynomials:

$$\text{Minimize } f_0(x) = \sum_{t=1}^{T_0} \sigma_{0t} c_{0t} \prod_{i=1}^N x_i^{a_{0it}}$$

subject to

$$f_j(x) = \sum_{t=1}^{T_j} \sigma_{jt} c_{jt} \prod_{i=1}^N x_i^{a_{jxit}} \leq \sigma_j, \quad j = 1, 2, \dots, J; \\ x_i > 0, \quad i = 1, 2, \dots, N.$$

Here, the parameter  $\sigma_{jt}$  and  $\sigma_j$  can take a value either 1 or -1. The corresponding dual problem can be written as follows:

$$\text{Maximize } Z(\delta, \sigma_0) = \sigma_0 \prod_{j=0}^J \prod_{t=1}^{T_j} \left( \frac{c_{jt} u_j}{\delta_{jt}} \right)^{\sigma_0 \sigma_{jt} \delta_{jt}}$$

subject to

$$u_0 = \sigma_0 \sum_{t=1}^{T_0} \sigma_{0t} \delta_{0t} = 1, \\ \sum_{j=0}^J \sum_{t=1}^{T_j} a_{jxit} \delta_{jt} \sigma_{jt} = 0, \quad i = 1, 2, \dots, N; \\ u_j = \sigma_j \sum_{t=1}^{T_j} \sigma_{jt} \delta_{jt} \geq 0, \quad j = 1, 2, \dots, J; \\ \delta_{jt} \geq 0, \quad j = 0, 1, 2, \dots, J, \quad t = 1, 2, \dots, T_j.$$

The binary variable  $\sigma_0$  can take only a 1 or a -1 subject to satisfaction of the above constraints. Since there is only one binary variable, two separate problems, one with  $\sigma_0 = 1$  and one with  $\sigma_0 = -1$ , can be solved. Thereafter, the solution with feasible values of dual variables can be accepted.

The above formulation introduces  $J$  extra inequality constraints with  $u_j \geq 0$ . This is required because for any inequality constraint, the condition  $\delta_{jt} \geq 0$  does not guarantee  $u_j \geq 0$ , which is a requirement at the optimal point. The dual problem is now required to be solved for all dual variables  $\delta_{jt}$  and an extra variable  $\sigma_0$ . Once the dual problem is solved, the corresponding primal variables can be found by solving the following simultaneous linear equations:

$$\sum_{i=1}^N a_{0it} \ln(x_i^*) = \ln \left( \frac{\sigma_0 \delta_{0t}^* f_0(x^*)}{c_{0t}} \right), \quad (5.9)$$

$$\sum_{i=1}^N a_{jxit} \ln(x_i^*) = \ln \left( \frac{\delta_{jt}^*}{u_j^* c_{jt}} \right), \quad j = 1, 2, \dots, J. \quad (5.10)$$

Both the above equations are valid for  $t = 1, 2, \dots, T_m$  and  $\delta_{0t}^* > 0$ . The degree of difficulty in this modified problem is

$$d = \sum_{j=0}^J T_j - N - 1.$$

We illustrate the working of this modified method by considering a simple exercise problem. The same GP algorithm outlined earlier can be used with the above modifications.

### EXERCISE 5.2.2

We would like to solve the following NLP problem:

$$\text{Minimize } f(x) = 10 - x_2$$

subject to

$$g(x) = 26 - (x_1 - 5)^2 - x_2^2 \geq 0, \quad x_1, x_2 > 0.$$

The constraint  $g(x)$  makes all points inside a circle of radius  $\sqrt{26}$  and centre at  $(5, 0)^T$  feasible. Since the objective is to minimize the function  $f(x) = 10 - x_2$ , the optimal solution is the point on the circle having maximum  $x_2$  value. This solution is  $(5.000, 5.099)^T$ . The optimal function value at this point is 4.901. We shall use the above primal-dual formulations to find this solution.

**Step 1** We first write the given problem in the primal form. We recognize that the constant term in the objective function does not affect the minimum solution and therefore can be omitted from the following discussion. Once the optimal function value is found, the constant can be added to obtain the true optimum value. By performing simple algebraic manipulation, we rewrite the above problem without the constant term in the objective function:

$$\text{Minimize } f_0(x) = -1(x_1)^0(x_2)^1$$

subject to

$$g(x) = (x_1)^2(x_2)^0 + (x_1)^0(x_2)^2 - 10(x_1)^1(x_2)^0 \leq 1, \quad x_1, x_2 > 0.$$

The above problem has two variables ( $N = 2$ ). The objective function has only one posynomial term ( $T_0 = 1$ ). In the above problem, there is only one constraint ( $J = 1$ ) and there are three posynomial terms in the constraint ( $T_1 = 3$ ). Thus, the degree of difficulty of the problem is  $(1 + 3 - 2 - 1)$  or 1. In the above problem, we observe the following parameter values:

$$\text{Objective function } \sigma_{01} = -1, \quad c_{01} = 1, \quad a_{011} = 0, \quad a_{021} = 1,$$

$$\text{Constraints: } \sigma_{11} = 1, \quad \sigma_{12} = 1, \quad c_{11} = 1,$$

$$a_{111} = 2, \quad a_{121} = 0, \quad \sigma_{13} = -1,$$

$$\sigma_1 = 1, \quad c_{12} = 1, \quad a_{112} = 0,$$

$$a_{122} = 2, \quad c_{13} = 10, \quad a_{113} = 1, \quad a_{123} = 0.$$

With these values, we form the dual problem:

$$\text{Maximize } Z(\delta, \sigma_0)$$

$$= \sigma_0 \left[ \left( \frac{1}{\delta_{01}} \right)^{-\delta_{01}} \left( \frac{u_1}{\delta_{11}} \right)^{\delta_{11}} \left( \frac{u_1}{\delta_{12}} \right)^{\delta_{12}} \left( \frac{10u_1}{\delta_{13}} \right)^{-\delta_{13}} \right]^{\sigma_0}$$

subject to

$$\begin{aligned} -\sigma_0 \delta_{01} &= 1, \\ 2\delta_{11} - \delta_{13} &= 0, \\ -\delta_{01} + 2\delta_{12} &= 0, \\ u_1 = \delta_{11} + \delta_{12} - \delta_{13} &\geq 0, \\ \delta_{01}, \delta_{11}, \delta_{12}, \delta_{13} &\geq 0. \end{aligned}$$

In general, the optimal solution to the above NLP problem can be obtained using one of the algorithms discussed in Chapter 4. Since the number of variables is less and equality constraints are linear, we use variable elimination method to find the optimal solution. From the first equality constraint, we notice that  $\sigma_0$  must be negative, because  $\delta_{01}$  has to be positive. Thus, we set  $\sigma_0 = -1$  in the remaining calculations. Since there are four variables and three equality constraints, the objective function may be written in terms of only one variable. From the first and third equality constraints, we observe that  $\delta_{01} = 1$  and  $\delta_{12} = 0.5$ . By writing the  $\delta_{13}$  in terms of  $\delta_{11}$ , we obtain the one-dimensional constrained problem:

$$\text{Maximize } Z(\delta_{11}) = -\frac{1}{\sqrt{2}}(0.04\delta_{11})^{-\delta_{11}}(0.5 - \delta_{11})^{(0.5-\delta_{11})}$$

subject to

$$\begin{aligned} \delta_{11} &\leq 0.5, \\ \delta_{11} &\geq 0. \end{aligned}$$

Using the golden section search method in the domain  $(0.0, 0.5)$ , we obtain the solution  $\delta_{11}^* = 0.4808$ . The other optimal dual variables are  $\delta_{12}^* = 0.5$ ,  $\delta_{13}^* = 0.9615$ ,  $u_1^* = 0.0192$ , and  $\sigma_0^* = -1$ .

**Step 2** Substituting the value of  $\delta_{11}^*$  in the above objective function, we obtain  $Z^* = -5.099$ . This value is also the optimal function value for the primal function:  $f_0^* = -5.099$ . Using Equations (5.9) and (5.10), we calculate the corresponding optimal primal variables:

$$\begin{aligned} x_2^* &= -f_0^*, & x_1^* &= \sqrt{\frac{\delta_{11}^*}{u_1^*}}, \\ x_2^* &= \sqrt{\frac{\delta_{12}^*}{u_1^*}}, & x_1^* &= \frac{\delta_{13}^*}{10u_1^*}. \end{aligned}$$

These equations yield the solution  $x_1^* = 5.000$  and  $x_2^* = 5.099$ . The true optimum function value is  $(f_0^* + 10)$  or 4.901.

### 5.3 Summary

Two special purpose optimization algorithms are described in this chapter. In addition to constraints associated with an optimization problem, many problems may have an additional restrictions of having discrete or integer variables. Most optimization algorithms discussed in the previous chapters may fail to correctly solve these problems because the underlying notion of search direction and gradient information at a particular point are local in nature. In discrete or integer programming problems, the points in the vicinity of a feasible point are not feasible. In this chapter, two different methods—a penalty function method and a branch-and-bound method—are described. In the penalty function method, all infeasible solutions corresponding to noninteger solutions of integer variables are penalized. The resulting penalty function is solved recursively to find the optimum. In the branch-and-bound method, the optimum solution is found by recursively partitioning (branching) the problem in several regions of the search space and by subsequent by fathoming (bounding) the nodes.

Since many engineering design problems (objective function and constraints) can be written as summation of several *posynomial* terms, the *geometric programming* (GP) method is found suitable and efficient in solving those problems (Beightler and Phillips, 1976; Duffin, et al., 1967). In the geometric programming method, the original (primal) problem is first transformed into an equivalent *dual* problem and solved. Thereafter, the optimal dual solution is transformed back to an equivalent primal solution. The GP method is efficient if the NLP problem can be written in the required primal form. Fortunately, many engineering problems can be written in the required primal form and, thus, GP methodology is popular in engineering design optimization.

## REFERENCES

- Beightler, C. S. and Phillips, D. T. (1976). *Applied Geometrical Programming*. New York: Wiley.
- Duffin, R. J., Peterson, E. L., and Zener, C. (1967). *Geometric Programming*. New York: Wiley.
- Rao, S. S. (1984). *Optimization Theory and Applications*. New Delhi: Wiley Eastern.
- Reklaitis, G. V., Ravindran, A., and Ragsdell, K. M. (1983). *Engineering Optimization—Methods and Applications*. New York: Wiley.
- Strang, G. (1980). *Linear Algebra and Its Applications*. Orlando: Academic Press.

## PROBLEMS

**5-1** In minimizing the mixed integer programming problem:

$$\text{Minimize } (x_2^2 + 2x_1 + 2)^4 + (x_5 - x_1)^2 + (x_3 - 3)^2 + x_4^4 + (x_2 + x_1 - 2)^2$$

subject to

$$2x_1 + x_2 - 3x_3^2 + 10 \geq 0,$$

$$x_1^2 - 2x_5 \geq 0,$$

$x_1, x_2, x_3$  integers,

we waive the integer restriction and solve the resulting NLP problem using penalty function method described in Chapter 4. The obtained minimum point is  $x = (1.9, 0.1, 3.2, 0, 2.0)^T$ . Find the true optimum solution by choosing suitable values for the integer variables.

**5-2** Perform two iterations of the penalty function method to minimize the following INLP problem:

$$\text{Minimize } x_1^4 + 3xy^2 + (10y + 1)^2$$

subject to

$$x \geq y,$$

$x, y$  integers.

**5-3** Perform two iterations of the branch-and-bound method to minimize the following INLP problem:

$$\text{Minimize } 0.1(x_1^4 + x_2^4) - x_1^2x_2$$

subject to

$$x_1^2 + 2x_2^2 \leq 100,$$

$x_1, x_2$  integers.

**5-4** Use the branch-and-bound method to find the optimal solution of the following NLP problem:

$$\text{Maximize } f(x, y) = x + 2y$$

subject to

$$x^2 + 7y \leq 49,$$

$x$  and  $y$  are integers.

- (i) Solve each intermediate NLP problem (NLP1, NLP2, ...) graphically (show one  $x$ - $y$  plot for each NLP). Branch with the variable having the highest infeasible value.

- (ii) Make a separate plot of the feasible region of each subproblem and show the corresponding optimal solution.
- (iii) Show a flowchart of all NLPs and their solutions with function values.

**5-5** Solve the following mixed integer program using the branch-and-bound algorithm.

$$\text{Maximize } 3x_1 + 2x_2$$

subject to

$$2x_1 + x_2 \leq 9,$$

$$-x_1 + 2x_2 \leq 4,$$

$$x_1 - x_2 \leq 3,$$

$$x_1, x_2 \geq 0,$$

$$x_1, x_2 \text{ integers.}$$

Show the progress of the algorithm by plotting feasible regions.

**5-6** Formulate a suitable primal problem of the following NLP problem:

$$\text{Minimize } (x_1 + 2x_2)^{3.5} / x_3 + x_2^2 x_3$$

subject to

$$2x_1 + 3x_2 x_3^2 + 4x_2 \leq 1,$$

$$x_2 \geq x_3^2 + 1,$$

$$x_1, x_2, x_3 > 0.$$

Write the corresponding dual problem. What is the degree of difficulty of the problem?

**5-7** Solve the following integer linear program using the branch-and-bound method:

$$\text{Minimize } x_1 + 10x_2,$$

subject to

$$66x_1 + 14x_2 \geq 1428,$$

$$-82x_1 + 28x_2 \geq 1306,$$

$$x_1, x_2 \text{ integer.}$$

Solve the intermediate linear programs graphically by clearly showing the feasible space and corresponding optimum solution in separate plots. Branch based on the largest absolute value of a non-integer variable. Maintain calculations up to three decimal places of accuracy. Also show a flowchart of branching and bounding procedure.

**5-8** Solve the following programs using the geometric programming method:

$$(i) \text{ Minimize } f(x_1, x_2) = (x_1^2 + 5x_2)^2 + \frac{100}{x_1 x_2}, \quad x_1, x_2 > 0.$$

$$(ii) \text{ Minimize } f(x_1, x_2, x_3)$$

$$= \frac{x_1 x_2}{x_3} + \frac{2x_1 x_3}{x_2} + \frac{x_2 x_3}{x_1} + \frac{1}{5x_1 x_2 x_3}, \quad x_1, x_2, x_3 > 0.$$

**5-9** Formulate a suitable primal problem of the following NLP problem:

$$\text{Minimize } x_1 x_3^2 + \exp(x_2)$$

subject to

$$x_1^2 + \exp(-2x_2) \leq 9,$$

$$x_2 < 0,$$

$$x_1, x_3 > 0.$$

Solve the equivalent dual problem.

**5-10** Solve the following NLP problem using the GP method:

$$\text{Minimize } x_1 x_2 x_3$$

subject to

$$\frac{x_1 x_2}{x_3} + \frac{x_1 x_3}{x_2} + \frac{x_2 x_3}{x_1} \leq 1,$$

$$x_1, x_2, x_3 > 0.$$

**5-11** Solve the following single-variable optimization problem using the GP method:

$$\text{Minimize } f(x) = x^3 - x, \quad x > 0.$$

How does the solution change if a constraint  $x \geq 0.7$  is added? Form the primal and dual problem in each case and solve.

**5-12** Solve the following problem using the GP method:

$$\text{Minimize } xy^2 - 3(y - 1)^2$$

subject to

$$x^2 - 6x + y \leq 0,$$

$$x \geq 3,$$

$$y \geq 2.$$

# 6

## Nontraditional Optimization Algorithms

---

This chapter describes two nontraditional search and optimization methods which are becoming popular in engineering optimization problems in the recent past. These algorithms are included in this book not because they are new but because they are found to be potential search and optimization algorithms for complex engineering optimization problems. *Genetic algorithms* (GAs) mimic the principles of natural genetics and natural selection to constitute search and optimization procedures. *Simulated annealing* mimics the cooling phenomenon of molten metals to constitute a search procedure. Since both these algorithms are abstractions from a natural phenomenon, they are very different search methods than those described in the previous chapters. We describe genetic algorithms first, followed by the simulated annealing procedure.

### 6.1 Genetic Algorithms

Genetic algorithms are computerized search and optimization algorithms based on the mechanics of natural genetics and natural selection. Professor John Holland of the University of Michigan, Ann Arbor envisaged the concept of these algorithms in the mid-sixties and published his seminal work (Holland, 1975). Thereafter, a number of his students and other researchers have contributed to developing this field. To date, most of the GA studies are available through a few books (Davis, 1991; Goldberg, 1989; Holland, 1975; Michalewicz, 1992) and through a number of international conference proceedings (Belew and Booker, 1991; Forrest, 1993; Grefenstette, 1985, 1987; Rawlins, 1991; Schaffer, 1989; Whitley, 1993). An extensive list of GA-related papers is referenced elsewhere (Goldberg, et al., 1992). GAs are fundamentally different than classical optimization algorithms we have

discussed in Chapters 2 through 5. We begin the discussion of GAs by first outlining the working principles of GAs and then highlighting the differences GAs have with the traditional search methods. Thereafter, we show a computer simulation to illustrate the working of GAs.

### 6.1.1 Working Principles

To illustrate the working principles of GAs, we first consider an unconstrained optimization problem. Later, we shall discuss how GAs can be used to solve a constrained optimization problem. Let us consider the following maximization problem:

$$\text{Maximize } f(x), \quad x_i^{(L)} \leq x_i \leq x_i^{(U)}, \quad i = 1, 2, \dots, N.$$

Although a maximization problem is considered here, a minimization problem can also be handled using GAs. The working of GAs is completed by performing the following tasks:

#### Coding

In order to use GAs to solve the above problem, variables  $x_i$ 's are first coded in some string structures. It is important to mention here that the coding of the variables is not absolutely necessary. There exist some studies where GAs are directly used on the variables themselves, but here we shall ignore the exceptions and discuss the working principle of a simple genetic algorithm. Binary-coded strings having 1's and 0's are mostly used. The length of the string is usually determined according to the desired solution accuracy. For example, if four bits are used to code each variable in a two-variable function optimization problem, the strings (0000 0000) and (1111 1111) would represent the points

$$(x_1^{(L)}, x_2^{(L)})^T \quad (x_1^{(U)}, x_2^{(U)})^T,$$

respectively, because the substrings (0000) and (1111) have the minimum and the maximum decoded values. Any other eight-bit string can be found to represent a point in the search space according to a fixed mapping rule. Usually, the following linear mapping rule is used:

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^{\ell_i} - 1} \text{ decoded value } (s_i). \quad (6.1)$$

In the above equation, the variable  $x_i$  is coded in a substring  $s_i$  of length  $\ell_i$ . The decoded value of a binary substring  $s_i$  is calculated as  $\sum_{i=0}^{\ell_i-1} 2^i s_i$ , where  $s_i \in (0, 1)$  and the string  $s$  is represented as  $(s_{\ell-1}s_{\ell-2}\dots s_2s_1s_0)$ . For example, a four-bit string (0111) has a decoded value equal to  $((1)2^0 + (1)2^1 + (1)2^2 + (0)2^3)$  or 7. It is worthwhile to mention here that with four bits to code each variable, there are only  $2^4$  or 16 distinct substrings

possible, because each bit-position can take a value either 0 or 1. The accuracy that can be obtained with a four-bit coding is only approximately 1/16th of the search space. But as the string length is increased by one, the obtainable accuracy increases exponentially to 1/32th of the search space. It is not necessary to code all variables in equal substring length. The length of a substring representing a variable depends on the desired accuracy in that variable. Generalizing this concept, we may say that with an  $\ell_i$ -bit coding for a variable, the obtainable accuracy in that variable is approximately  $(x_i^{(U)} - x_i^{(L)})/2^{\ell_i}$ . Once the coding of the variables has been done, the corresponding point  $x = (x_1, x_2, \dots, x_N)^T$  can be found using Equation (6.1). Thereafter, the function value at the point  $x$  can also be calculated by substituting  $x$  in the given objective function  $f(x)$ .

### Fitness function

As pointed out earlier, GAs mimic the survival-of-the-fittest principle of nature to make a search process. Therefore, GAs are naturally suitable for solving maximization problems. Minimization problems are usually transformed into maximization problems by some suitable transformation. In general, a *fitness* function  $\mathcal{F}(x)$  is first derived from the objective function and used in successive genetic operations. Certain genetic operators require that the fitness function be nonnegative, although certain operators do not have this requirement. For maximization problems, the fitness function can be considered to be the same as the objective function or  $\mathcal{F}(x) = f(x)$ . For minimization problems, the fitness function is an equivalent maximization problem chosen such that the optimum point remains unchanged. A number of such transformations are possible. The following fitness function is often used:

$$\mathcal{F}(x) = 1/(1 + f(x)). \quad (6.2)$$

This transformation does not alter the location of the minimum, but converts a minimization problem to an equivalent maximization problem. The fitness function value of a string is known as the string's *fitness*.

The operation of GAs begins with a population of random strings representing design or decision variables. Thereafter, each string is evaluated to find the fitness value. The population is then operated by three main operators—*reproduction*, *crossover*, and *mutation*—to create a new population of points. The new population is further evaluated and tested for termination. If the termination criterion is not met, the population is iteratively operated by the above three operators and evaluated. This procedure is continued until the termination criterion is met. One cycle of these operations and the subsequent evaluation procedure is known as a *generation* in GA's terminology. The operators are described next.

### GA operators

Reproduction is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool.

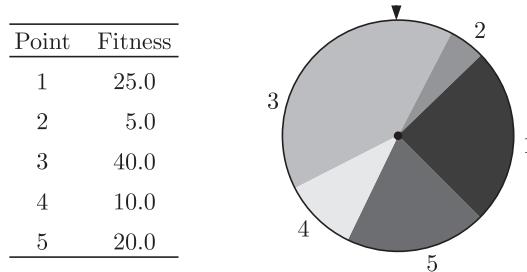
That is why the reproduction operator is sometimes known as the selection operator. There exist a number of reproduction operators in GA literature, but the essential idea in all of them is that the above-average strings are picked from the current population and their multiple copies are inserted in the mating pool in a probabilistic manner. The commonly-used reproduction operator is the proportionate reproduction operator where a string is selected for the mating pool with a probability proportional to its fitness. Thus, the  $i$ -th string in the population is selected with a probability proportional to  $\mathcal{F}_i$ . Since the population size is usually kept fixed in a simple GA, the sum of the probability of each string being selected for the mating pool must be one. Therefore, the probability for selecting the  $i$ -th string is

$$p_i = \frac{\mathcal{F}_i}{\sum_{j=1}^n \mathcal{F}_j},$$

where  $n$  is the population size. One way to implement this selection scheme is to imagine a roulette-wheel with its circumference marked for each string proportionate to the string's fitness. The roulette-wheel is spun  $n$  times, each time selecting an instance of the string chosen by the roulette-wheel pointer. Since the circumference of the wheel is marked according to a string's fitness, this roulette-wheel mechanism is expected to make  $\mathcal{F}_i/\bar{\mathcal{F}}$  copies of the  $i$ -th string in the mating pool. The average fitness of the population is calculated as

$$\bar{\mathcal{F}} = \sum_{i=1}^n \mathcal{F}_i/n.$$

Figure 6.1 shows a roulette-wheel for five individuals having different fitness values. Since the third individual has a higher fitness value than any other, it



**Figure 6.1** A roulette-wheel marked for five individuals according to their fitness values. The third individual has a higher probability of selection than any other.

is expected that the roulette-wheel selection will choose the third individual more than any other individual. This roulette-wheel selection scheme can be simulated easily. Using the fitness value  $\mathcal{F}_i$  of all strings, the probability of selecting a string  $p_i$  can be calculated. Thereafter, the cumulative probability

$(P_i)$  of each string being copied can be calculated by adding the individual probabilities from the top of the list. Thus, the bottom-most string in the population should have a cumulative probability  $(P_n)$  equal to 1. The roulette-wheel concept can be simulated by realizing that the  $i$ -th string in the population represents the cumulative probability values from  $P_{i-1}$  to  $P_i$ . The first string represents the cumulative values from zero to  $P_1$ . Thus, the cumulative probability of any string lies between 0 to 1. In order to choose  $n$  strings,  $n$  random numbers between zero to one are created at random. Thus, a string that represents the chosen random number in the cumulative probability range (calculated from the fitness values) for the string is copied to the mating pool. This way, the string with a higher fitness value will represent a larger range in the cumulative probability values and therefore has a higher probability of being copied into the mating pool. On the other hand, a string with a smaller fitness value represents a smaller range in cumulative probability values and has a smaller probability of being copied into the mating pool. We illustrate the working of this roulette-wheel simulation later through a computer simulation of GAs.

In reproduction, good strings in a population are probabilistically assigned a larger number of copies and a mating pool is formed. It is important to note that no new strings are formed in the reproduction phase. In the crossover operator, new strings are created by exchanging information among strings of the mating pool. Many crossover operators exist in the GA literature. In most crossover operators, two strings are picked from the mating pool at random and some portions of the strings are exchanged between the strings. A single-point crossover operator is performed by randomly choosing a crossing site along the string and by exchanging all bits on the right side of the crossing site as shown:

$$\begin{array}{cc|ccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{array} \Rightarrow \begin{array}{cc|ccc} 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{array}$$

The two strings participating in the crossover operation are known as parent strings and the resulting strings are known as children strings. It is intuitive from this construction that good substrings from parent strings can be combined to form a better child string, if an appropriate site is chosen. Since the knowledge of an appropriate site is usually not known beforehand, a random site is often chosen. With a random site, the children strings produced may or may not have a combination of good substrings from parent strings, depending on whether or not the crossing site falls in the appropriate place. But we do not worry about this too much, because if good strings are created by crossover, there will be more copies of them in the next mating pool generated by the reproduction operator. But if good strings are not created by crossover, they will not survive too long, because reproduction will select against those strings in subsequent generations.

It is clear from this discussion that the effect of crossover may be detrimental or beneficial. Thus, in order to preserve some of the good strings

that are already present in the mating pool, not all strings in the mating pool are used in crossover. When a crossover probability of  $p_c$  is used, only  $100p_c$  per cent strings in the population are used in the crossover operation and  $100(1 - p_c)$  per cent of the population remains as they are in the current population<sup>1</sup>.

A crossover operator is mainly responsible for the search of new strings, even though a mutation operator is also used for this purpose sparingly. The mutation operator changes 1 to 0 and vice versa with a small mutation probability,  $p_m$ . The bit-wise mutation is performed bit by bit by flipping a coin<sup>2</sup> with a probability  $p_m$ . If at any bit the outcome is true then the bit is altered; otherwise the bit is kept unchanged. The need for mutation is to create a point in the neighbourhood of the current point, thereby achieving a local search around the current solution. The mutation is also used to maintain diversity in the population. For example, consider the following population having four eight-bit strings:

```
0110 1011
0011 1101
0001 0110
0111 1100
```

Notice that all four strings have a 0 in the left-most bit position. If the true optimum solution requires 1 in that position, then neither reproduction nor crossover operator described above will be able to create 1 in that position. The inclusion of mutation introduces some probability ( $Np_m$ ) of turning 0 into 1.

These three operators are simple and straightforward. The reproduction operator selects good strings and the crossover operator recombines good substrings from good strings together to hopefully create a better substring. The mutation operator alters a string locally to hopefully create a better string. Even though none of these claims are guaranteed and/or tested while creating a string, it is expected that if bad strings are created they will be eliminated by the reproduction operator in the next generation and if good strings are created, they will be increasingly emphasized. Interested readers may refer to Goldberg (1989) and other GA literature given in the references for further insight and some mathematical foundations of genetic algorithms.

Here, we outline some differences and similarities of GAs with traditional optimization methods.

---

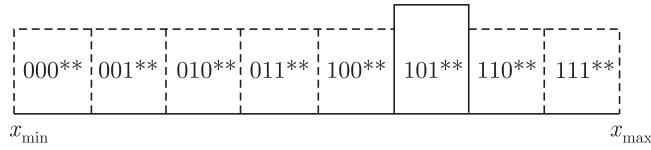
<sup>1</sup>Even though the best  $(1 - p_c)100\%$  of the current population can be copied deterministically to the new population, this is usually performed at random.

<sup>2</sup>Flipping of a coin with a probability  $p$  is simulated as follows. A number between 0 to 1 is chosen at random. If the random number is smaller than  $p$ , the outcome of coin-flipping is true, otherwise the outcome is false.

### 6.1.2 Differences between GAs and Traditional Methods

As seen from the above description of the working principles of GAs, they are radically different from most of the traditional optimization methods described in Chapters 2 to 4. However, the fundamental differences are described in the following paragraphs.

GAs work with a string-coding of variables instead of the variables. The advantage of working with a coding of variables is that the coding discretizes the search space, even though the function may be continuous. On the other hand, since GAs require only function values at various discrete points, a discrete or discontinuous function can be handled with no extra cost. This allows GAs to be applied to a wide variety of problems. Another advantage is that the GA operators exploit the similarities in string-structures to make an effective search. Let us discuss this important aspect of GAs in somewhat more details. A *schema* (pl. *schemata*) represents a number of strings with similarities at certain string positions. For example, in a five-bit problem, the schema (101\*\*) (a \* denotes either a 0 or a 1) represents four strings (10100), (10101), (10110), and (10111). In the decoded parameter space, a schema represents a continuous or discontinuous region in the search space. Figure 6.2 shows that the above schema represents one-eighth of the search space. Since in an  $\ell$ -bit schema, every position can take either 0, 1, or \*, there



**Figure 6.2** A schema with three fixed positions divides the search space into eight regions. The schema (101\*\*) is highlighted.

are a total of  $3^\ell$  schemata possible. A finite population of size  $n$  contains only  $n$  strings, but contains many schemata. Goldberg (1989) has shown that due to the action of GA operators, the number of strings  $m(H, t)$  representing a schema  $H$  at any generation  $t$  grows to a number  $m(H, t + 1)$  in the next generation as follows:

$$m(H, t + 1) \geq m(H, t) \underbrace{\frac{\mathcal{F}(H)}{\mathcal{F}} \left[ 1 - p_c \frac{\delta(H)}{\ell - 1} - p_m o(H) \right]}_{\text{growth factor, } \phi}, \quad (6.3)$$

where  $\mathcal{F}(H)$  is the fitness of the schema  $H$  calculated by averaging the fitness of all strings representing the schema,  $\delta(H)$  is the defining length of the schema  $H$  calculated as the difference in the outermost defined positions, and  $o(H)$  is the order of the schema  $H$  calculated as the number of fixed positions in the schema. For example, the schema  $H = 101**$  has a defining length equal to  $\delta(H) = 3 - 1 = 2$  and has an order  $o(H) = 3$ . The growth factor  $\phi$  defined in the above equation can be greater than, less than, or

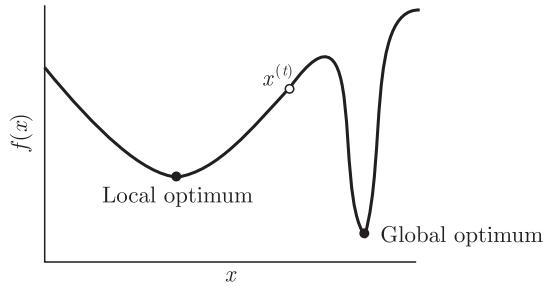
equal to 1 depending on the schema  $H$  and the chosen GA parameters. If for a schema the growth factor  $\phi \geq 1$ , the number of strings representing that schema grows with generation, otherwise the representative strings of the schema reduce with generation. The above inequality suggests that the schema having a small defining length (small  $\delta(H)$ ), a few fixed positions (small  $o(H)$ ), and above-average fitness ( $\mathcal{F}(H) > \bar{\mathcal{F}}$ ), the growth factor  $\phi$  is likely to be greater than 1. Schemata for which the growth factor is greater than 1 grows exponentially with generation. These schemata usually represent a large, good region (a region with many high fitness points) in the search space. These schemata are known as *building blocks* in GA parlance. These building blocks representing different good regions in the search space get exponentially more copies and get combined with each other by the action of GA operators and finally form the optimum or a near-optimum solution. Even though this is the basic understanding of how GAs work, there exists some mathematical rigor to this hypothesis (Davis and Principe, 1991; Vose and Liepins, 1991). Holland (1975) has shown that even though  $n$  population members are modified in a generation, about  $n^3$  schemata get processed in a generation. This leverage comes without any extra book-keeping (Goldberg, 1989) and provides an *implicit parallelism* in the working of genetic algorithms. Even though there are a number of advantages of using a coding of variables, there are also some disadvantages. One of the drawbacks of using a coding representation is that a meaningful and an appropriate coding of the problem needs to be used, otherwise GAs may not converge to the right solution (Goldberg et al., 1989; Kargupta et al., 1992). However, a general guideline would be to use a coding that does not make the problem harder than the original problem.

The most striking difference between GAs and many traditional optimization methods is that GAs work with a population of points instead of a single point. Because there are more than one string being processed simultaneously, it is very likely that the expected GA solution may be a global solution. Even though some traditional algorithms are population-based, like Box's evolutionary optimization and complex search methods, those methods do not use previously obtained information efficiently. In GAs, previously found good information is emphasized using reproduction operator and propagated adaptively through crossover and mutation operators. Another advantage with a population-based search algorithm is that multiple optimal solutions can be captured in the population easily, thereby reducing the effort to use the same algorithm many times. Some extensions of GAs along these directions—multimodal function optimization (Deb, 1989; Goldberg and Richardson, 1987) and multiobjective function optimization (Horn and Nafpliotis, 1993; Schaffer, 1984; Srinivas and Deb, 1995)—have been researched and are outlined in Section 6.1.8.

In discussing GA operators or their working principles in the previous section, nothing has been mentioned about the gradient or any other auxiliary problem information. In fact, GAs do not require any auxiliary information

except the objective function values. Although the direct search methods used in traditional optimization methods do not explicitly require the gradient information, some of those methods use search directions that are similar in concept to the gradient of the function. Moreover, some direct search methods work under the assumption that the function to be optimized is unimodal and continuous. In GAs, no such assumption is necessary.

One other difference in the operation of GAs is the use of probabilities in their operators. None of the genetic operators work deterministically. In the reproduction operator, even though a string is expected to have  $\mathcal{F}_i/\bar{\mathcal{F}}$  copies in the mating pool, a simulation of the roulette-wheel selection scheme is used to assign the true number of copies. In the crossover operator, even though good strings (obtained from the mating pool) are crossed, strings to be crossed are created at random and cross-sites are created at random. In the mutation operator, a random bit is suddenly altered. The action of these operators may appear to be naive, but careful studies may provide some interesting insights about this type of search. The basic problem with most of the traditional methods is that they use fixed transition rules to move from one point to another. For instance, in the steepest descent method, the search direction is always calculated as the negative of the gradient at any point, because in that direction the reduction in the function value is maximum. In trying to solve a multimodal problem with many local optimum points (interestingly, many real-world engineering optimization problems are likely to be multimodal), search procedures may easily get trapped in one of the local optimum points. Consider the bimodal function shown in Figure 6.3. The objective function has one local minimum and one global minimum. If



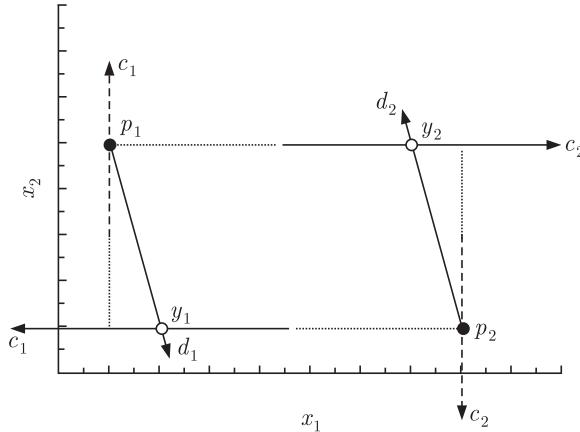
**Figure 6.3** An objective function with one local optimum and one global optimum. The point  $x^{(t)}$  is in the local basin.

the initial point is chosen to be a point in the local basin (point  $x^{(t)}$  in the figure), the steepest descent algorithm will eventually find the local optimum point. Since the transition rules are rigid, there is no escape from these local optima. The only way to solve the above problem to global optimality is to have a starting point in the global basin. Since this information is usually not known in any problem, the steepest-descent method (and for that matter most traditional methods) fails to locate the global optimum. We show simulation results showing inability of the steepest descent method to find the

global optimum point on a multimodal problem in Section 6.3. However, these traditional methods can be best applied to a special class of problems suitable for those methods. For example, the gradient search methods will outperform almost any algorithm in solving continuous, unimodal problems, but they are not suitable for multimodal problem. Thus, in general, traditional methods are not robust. A robust algorithm can be designed in such a way that it uses the steepest descent direction most of the time, but also uses the steepest ascent direction (or any other direction) with some probability. Such a mixed strategy may require more number of function evaluations to solve continuous, unimodal problems, because of the extra computations involved in trying with non-descent directions. But this strategy may be able to solve complex, multimodal problems to global optimality. In the multimodal problem, shown in the above figure, the mixed strategy may take the point  $x^{(t)}$  into the global basin (when tried with non-descent directions) and finally find the global optimum point. GAs use similar search strategies by using probability in all their operators. Since an initial random population is used, to start with, the search can proceed in any direction and no major decisions are made in the beginning. Later on, when the population begins to converge in some bit positions, the search direction narrows and a near-optimal solution is achieved. This nature of narrowing the search space as the search progresses is adaptive and is a unique characteristic of genetic algorithms.

### 6.1.3 Similarities between GAs and Traditional Methods

Even though GAs are different than most traditional search algorithms, there are some similarities. In traditional search methods, where a search direction is used to find a new point, at least two points are either implicitly or explicitly used to define the search direction. In the Hooke-Jeeves pattern search method, a pattern move is created using two points. In gradient-based methods, the search direction requires derivative information which is usually calculated using function values at two neighbouring points. In the crossover operator (which is mainly responsible for the GA search), two points are also used to create two new points. Thus, the crossover operation is similar to a directional search method except that the search direction is not fixed for all points in the population and that no effort is made to find the optimal point in any particular direction. Consider a two-variable optimization problem shown in Figure 6.4, where two parent points  $p_1$  and  $p_2$  are participated in the crossover. Under the single-point crossover operator, one of the two substrings is crossed. It can be shown that the two children points can only lie along directions ( $c_1$  and  $c_2$ ) shown in the figure (either along solid arrows or along dashed arrows). The exact locations of the children points along these directions depend on the relative distance between the parents (Deb and Agrawal, 1995). The points  $y_1$  and  $y_2$  are the two typical children points obtained after crossing the parent points  $p_1$  and  $p_2$ . Thus, it may be envisaged that point  $p_1$  has moved in the direction from  $d_1$  up to the point  $y_1$  and similarly the point  $p_2$  has moved to the point  $y_2$ .



**Figure 6.4** The action of a single-point crossover operator on a two-variable search space. The points  $p_1$  and  $p_2$  are parent points and  $c_1$  and  $c_2$  are children points.

Since the two points used in the crossover operator are chosen at random, many such search directions are possible. Among them some directions may lead to the global basin and some directions may not. The reproduction operator has an indirect effect of filtering the good search directions and help guide the search. The purpose of the mutation operator is to create a point in the vicinity of the current point. The search in the mutation operator is similar to a local search method such as the exploratory search used in the Hooke-Jeeves method. With the discussion of the differences and similarities of GAs with traditional methods, we are now ready to present the algorithm in a step-by-step format.

### Algorithm

**Step 1** Choose a coding to represent problem parameters, a selection operator, a crossover operator, and a mutation operator. Choose population size,  $n$ , crossover probability,  $p_c$ , and mutation probability,  $p_m$ . Initialize a random population of strings of size  $\ell$ . Choose a maximum allowable generation number  $t_{\max}$ . Set  $t = 0$ .

**Step 2** Evaluate each string in the population.

**Step 3** If  $t > t_{\max}$  or other termination criteria is satisfied, **Terminate**.

**Step 4** Perform reproduction on the population.

**Step 5** Perform crossover on random pairs of strings.

**Step 6** Perform mutation on every string.

**Step 7** Evaluate strings in the new population. Set  $t = t + 1$  and go to Step 3.

The algorithm is straightforward with repeated application of three operators (Steps 4 to 7) to a population of points. We show the working of this algorithm to the unconstrained Himmelblau function used in Chapter 3.

### EXERCISE 6.1.1

The objective is to minimize the function

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

in the interval  $0 \leq x_1, x_2 \leq 6$ . Recall that the true solution to this problem is  $(3, 2)^T$  having a function value equal to zero.

**Step 1** In order to solve this problem using genetic algorithms, we choose binary coding to represent variables  $x_1$  and  $x_2$ . In the calculations here, 10 bits are chosen for each variable, thereby making the total string length equal to 20. With 10 bits, we can get a solution accuracy of  $(6 - 0)/(2^{10} - 1)$  or 0.006 in the interval  $(0, 6)$ . We choose roulette-wheel selection, a single-point crossover, and a bit-wise mutation operator. The crossover and mutation probabilities are assigned to be 0.8 and 0.05, respectively. We decide to have 20 points in the population. The random population created using Knuth's (1981) random number generator<sup>3</sup> with a random seed equal to 0.760 is shown in Table 6.1. We set  $t_{\max} = 30$  and initialize the generation counter  $t = 0$ .

**Step 2** The next step is to evaluate each string in the population. We calculate the fitness of the first string. The first substring (1100100000) decodes to a value equal to  $(2^9 + 2^8 + 2^5)$  or 800. Thus, the corresponding parameter value is equal to  $0 + (6 - 0) \times 800/1023$  or 4.692. The second substring (1110010000) decodes to a value equal to  $(2^9 + 2^8 + 2^7 + 2^4)$  or 912. Thus, the corresponding parameter value is equal to  $0 + (6 - 0) \times 912/1023$  or 5.349. Thus, the first string corresponds to the point  $x^{(1)} = (4.692, 5.349)^T$ . These values can now be substituted in the objective function expression to obtain the function value. It is found that the function value at this point is equal to  $f(x^{(1)}) = 959.680$ . We now calculate the fitness function value at this point using the transformation rule:  $\mathcal{F}(x^{(1)}) = 1.0/(1.0 + 959.680) = 0.001$ . This value is used in the reproduction operation. Similarly, other strings in the population are evaluated and fitness values are calculated. Table 6.1 shows the objective function value and the fitness value for all 20 strings in the initial population.

**Step 3** Since  $t = 0 < t_{\max} = 30$ , we proceed to Step 4.

**Step 4** At this step, we select good strings in the population to form the mating pool. In order to use the roulette-wheel selection procedure, we first calculate the average fitness of the population. By adding the fitness values of all strings and dividing the sum by the population size, we obtain  $\bar{\mathcal{F}} = 0.008$ .

---

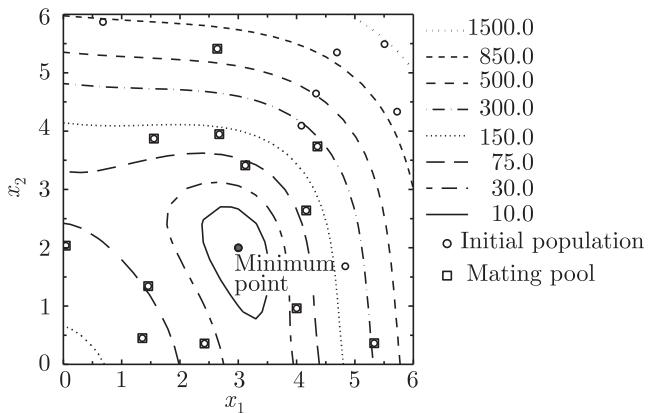
<sup>3</sup>A FORTRAN code implementing the random number generator appears in the GA code presented at the end of this chapter.

Table 6.1 Evaluation and Reproduction Phases on a Random Population

String	Mating pool													
	Substring-2	Substring-1	$x_2$	$x_1$	$f(x)$	$\mathcal{F}(x)$	$A$	$B$	$C$	$D$	$E$	$F$	Substring-2	Substring-1
1 1110010000 1100100000 5.349 4.692 959.680 0.001 0.13 0.007 0.007 0.472 10 0 0010100100 1010101010														
2 0001001101 0011100111 0.452 1.355 105.520 0.009 1.10 0.055 0.062 0.108 3 1 1010100001 0111001000														
3 1010100001 0111001000 3.947 2.674 126.685 0.008 0.98 0.049 0.111 0.045 2 1 0001001101 0111001111														
4 1001000110 1000010100 3.413 3.120 65.026 0.015 1.85 0.093 0.204 0.723 14 2 1110011011 0111000010														
5 1100011000 1011100011 4.645 4.334 512.197 0.002 0.25 0.013 0.217 0.536 10 0 0010100100 1010101010														
6 0011100101 0011111000 1.343 1.455 70.868 0.014 1.71 0.086 0.303 0.931 19 2 0011100010 1011000011														
7 0101011011 0000000111 2.035 0.041 88.273 0.011 1.34 0.067 0.370 0.972 19 1 0011100010 1011000011														
8 1110101000 1110101011 5.490 5.507 1436.563 0.001 0.12 0.006 0.376 0.817 17 0 0111000010 1011000110														
9 1001111101 1011100111 3.736 4.358 265.556 0.004 0.49 0.025 0.401 0.363 7 1 0101110111 0000000111														
10 0010100100 1010101010 0.962 4.000 39.849 0.024 2.96 0.148 0.549 0.189 4 3 1001000110 1000010100														
11 1111101001 0001110100 5.871 0.680 814.117 0.001 0.14 0.007 0.556 0.220 6 0 0011100101 0011111000														
12 0000111101 0110011101 0.358 2.422 42.598 0.023 2.84 0.142 0.698 0.288 6 3 0011100101 0011111000														
13 0000111110 1110001101 0.364 5.331 318.746 0.003 0.36 0.018 0.716 0.615 12 1 0000111101 0110011101														
14 1110011011 0111000010 5.413 2.639 624.164 0.002 0.24 0.012 0.728 0.712 13 1 0000111101 0110011101														
15 1010111010 1010111000 4.094 4.082 286.800 0.003 0.37 0.019 0.747 0.607 12 0 0000111101 0110011101														
16 0100011111 1100111000 1.683 4.833 197.556 0.005 0.61 0.030 0.777 0.192 4 0 1001000110 1000010100														
17 0111000010 1011000110 2.639 4.164 97.699 0.010 1.22 0.060 0.837 0.386 9 1 100111101 1011100111														
18 1010010100 0100001001 3.871 1.554 113.201 0.009 1.09 0.054 0.891 0.872 18 1 1010010100 0100001001														
19 0011100010 1011000011 1.326 4.147 57.753 0.017 2.08 0.103 0.994 0.589 12 2 0000111101 0110011101														
20 1011100011 1111010000 4.334 5.724 987.955 0.001 0.13 0.006 1.000 0.413 10 0 0010100100 1010101010														

$A$  : Expected count       $C$  : Cumulative probability of selection       $E$  : String number  
 $B$  : Probability of selection       $D$  : Random number between 0 and 1       $F$  : True count in the mating pool

The next step is to compute the expected count of each string as  $\mathcal{F}(x)/\bar{\mathcal{F}}$ . The values are calculated and shown in column A of Table 6.1. In other words, we can compute the probability of each string being copied in the mating pool by dividing these numbers with the population size (column B). Once these probabilities are calculated, the cumulative probability can also be computed. These distributions are also shown in column C of Table 6.1. In order to form the mating pool, we create random numbers between zero and one (given in column D) and identify the particular string which is specified by each of these random numbers. For example, if the random number 0.472 is created, the tenth string gets a copy in the mating pool, because that string occupies the interval (0.401, 0.549), as shown in column C. Column E refers to the selected string. Similarly, other strings are selected according to the random numbers shown in column D. After this selection procedure is repeated  $n$  times ( $n$  is the population size), the number of selected copies for each string is counted. This number is shown in column F. The complete mating pool is also shown in the table. Columns A and F reveal that the theoretical expected count and the true count of each string more or less agree with each other. Figure 6.5 shows the initial random population and the mating pool after reproduction. The



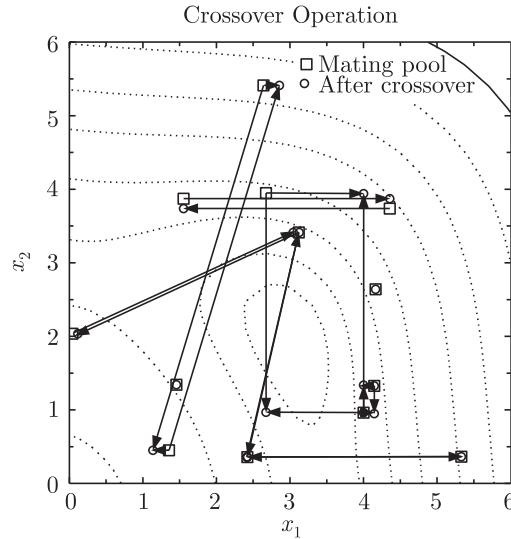
**Figure 6.5** The initial population (marked with empty circles) and the mating pool (marked with boxes) on a contour plot of the objective function. The best point in the population has a function value 39.849 and the average function value of the initial population is 360.540.

points marked with an enclosed box are the points in the mating pool. The action of the reproduction operator is clear from this plot. The inferior points have been probabilistically eliminated from further consideration. Notice that not all selected points are better than all rejected points. For example, the 14th individual (with a fitness value 0.002) is selected but the 16th individual (with a function value 0.005) is not selected.

Although the above roulette-wheel selection is easier to implement, it is noisy. A more stable version of this selection operator is sometimes used. After the expected count for each individual string is calculated, the strings

are first assigned copies exactly equal to the mantissa of the expected count. Thereafter, the regular roulette-wheel selection is implemented using the decimal part of the expected count as the probability of selection. This selection method is less noisy and is known as the *stochastic remainder* selection.

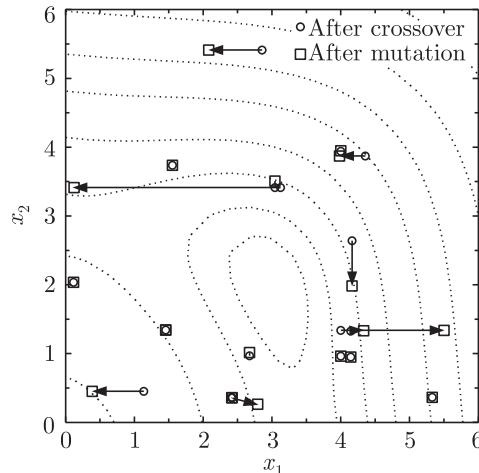
**Step 5** At this step, the strings in the mating pool are used in the crossover operation. In a single-point crossover, two strings are selected at random and crossed at a random site. Since the mating pool contains strings at random, we pick pairs of strings from the top of the list. Thus, strings 3 and 10 participate in the first crossover operation. When two strings are chosen for crossover, first a coin is flipped with a probability  $p_c = 0.8$  to check whether a crossover is desired or not. If the outcome of the coin-flipping is true, the crossing over is performed, otherwise the strings are directly placed in an intermediate population for subsequent genetic operation. It turns out that the outcome of the first coin-flipping is true, meaning that a crossover is required to be performed. The next step is to find a cross-site at random. We choose a site by creating a random number between  $(0, \ell - 1)$  or  $(0, 19)$ . It turns out that the obtained random number is 11. Thus, we cross the strings at the site 11 and create two new strings. After crossover, the children strings are placed in the intermediate population. Then, strings 14 and 2 (selected at random) are used in the crossover operation. This time the coin-flipping comes true again and we perform the crossover at the site 8 found at random. The new children strings are put into the intermediate population. Figure 6.6 shows how points cross over and form new points. The points marked with a small



**Figure 6.6** The population after the crossover operation. Two points are crossed over to form two new points. Of ten pairs of strings, seven pairs are crossed.

box are the points in the mating pool and the points marked with a small circle are children points created after crossover operation. Notice that not all 10 pairs of points in the mating pool cross with each other. With the flipping of a coin with a probability  $p_c = 0.8$ , it turns out that fourth, seventh, and tenth crossovers come out to be false. Thus, in these cases, the strings are copied directly into the intermediate population. The complete population at the end of the crossover operation is shown in Table 6.2. It is interesting to note that with  $p_c = 0.8$ , the expected number of crossover in a population of size 20 is  $0.8 \times 20/2$  or 8. In this exercise problem, we performed seven crossovers and in three cases we simply copied the strings to the intermediate population. Figure 6.6 shows that some good points and some not-so-good points are created after crossover. In some cases, points far away from the parent points are created and in some cases points close to the parent points are created.

**Step 6** The next step is to perform mutation on strings in the intermediate population. For bit-wise mutation, we flip a coin with a probability  $p_m = 0.05$  for every bit. If the outcome is true, we alter the bit to 1 or 0 depending on the bit value. With a probability of 0.05, a population size 20, and a string length 20, we can expect to alter a total of about  $0.05 \times 20 \times 20$  or 20 bits in the population. Table 6.2 shows the mutated bits in bold characters in the table. As counted from the table, we have actually altered 16 bits. Figure 6.7 shows the effect of mutation on the intermediate population. In some cases,



**Figure 6.7** The population after mutation operation. Some points do not get mutated and remain unaltered. The best point in the population has a function value 18.886 and the average function value of the population is 140.210, an improvement of over 60 per cent.

the mutation operator changes a point locally and in some other it can bring a large change. The points marked with a small circle are points in the intermediate population. The points marked with a small box constitute the

**Table 6.1** Crossover and Mutation Operators

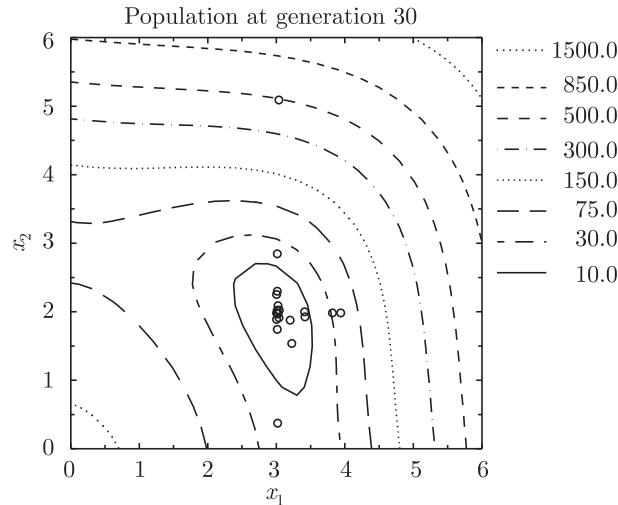
Matting pool	Substring-1	G	H	Intermediate population				Mutation			$\mathcal{F}(x)$		
				Substring-2	Substring-1	Substring-2	Substring-1	SubString-2	SubString-1	$x_1$			
0010100100	1010101010	Y	9	0010100101	0111001000	001010101010	011100100001	001010101010	011100100010	1.015	2.674	18.886	0.050
1010100001	0111001000	Y	9	1010100000	1010101010	1010100001	101010101010	3.947	4.000	238.322	0.004		
0001001101	0011100111	Y	12	0001001101	0011000010	0001001101	0001000010	0.452	0.387	149.204	0.007		
1110011011	0111000010	Y	12	1110011011	0111100111	1110011011	011011000111	5.413	2.082	596.340	0.002		
0010100100	1010101010	Y	5	0010100010	1011000011	0010100010	101100001111	0.950	4.147	54.851	0.018		
0011100010	1011000011	Y	5	00111000100	1010101010	00111000100	101010101010	1.337	5.501	424.583	0.002		
0011100010	1011000011	N		0011100010	1011000011	0011100011	10110000111	1.331	4.334	83.929	0.012		
0111000010	10110000110	N		0111000010	10110000110	0110101001	101100001110	1.982	4.164	70.472	0.014		
0101011011	00000000111	Y	14	0101011011	00000010100	0101011011	00000010100	2.035	0.117	87.633	0.011		
1001000110	10000010100	Y	14	1001000110	10000000111	1001010110	100000001111	3.507	3.044	72.789	0.014		
0011100101	0011111000	Y	1	0011100101	0011111000	0011100101	0011111000	1.343	1.455	70.868	0.014		
0011100101	0011111000	Y	1	0011100101	0011111000	0011100101	0011111000	1.343	1.455	70.868	0.014		
0000111101	0110011101	N		0000111101	0110011101	0000111101	0111011010	0.264	2.792	25.783	0.037		
0000111110	1110001101	N		0000111110	1110001101	0000111110	1110001101	0.364	5.331	318.746	0.003		
00000111101	0110011101	Y	18	00000111101	0110011100	00000111101	0110011100	0.358	2.416	42.922	0.023		
1001000110	10000010100	Y	18	1001000110	10000010101	1001000110	0000010101	3.413	0.123	80.127	0.012		
1001111101	1011100111	Y	10	1001111101	0100001001	1001111101	0100001001	3.736	1.554	95.968	0.010		
1010010100	0100001001	Y	10	1010010100	1011100111	1010010100	1010011011	3.871	3.982	219.426	0.005		
0000111101	0110011101	N		0000111101	0110011101	0000111101	0110011101	0.358	2.422	42.598	0.023		
0010100100	1010101010	N		0010100100	1010101010	0010100100	1010101010	0.962	4.000	39.849	0.024		

 $G$  : Whether crossover (Y yes, N no), $H$  : Crossing site

new population (obtained after reproduction, crossover, and mutation). It is interesting to note that if only one bit is mutated in a string, the point is moved along a particular variable only. Like the crossover operator, the mutation operator has created some points better and some points worse than the original points. This flexibility enables GA operators to explore the search space properly before converging to a region prematurely. Although this requires some extra computation, this flexibility is essential to solve global optimization problems.

**Step 7** The resulting population becomes the new population. We now evaluate each string as before by first identifying the substrings for each variable and mapping the decoded values of the substrings in the chosen intervals. This completes one iteration of genetic algorithms. We increment the generation counter to  $t = 1$  and proceed to Step 3 for the next iteration. The new population after one iteration of GAs is shown in Figure 6.7 (marked with empty boxes). The figure shows that in one iteration, some good points have been found. Table 6.2 also shows the fitness values and objective function values of the new population members.

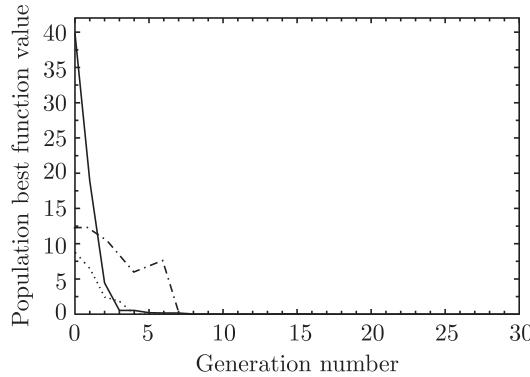
The average fitness of the new population is calculated to be 0.015, a remarkable improvement from that in the initial population (recall that the average in the initial population was 0.008). The best point in this population is found to have a fitness equal to 0.050, which is also better than that in the initial population (0.024). This process continues until the maximum allowable generation is reached or some other termination criterion is met. The population after 25 generation is shown in Figure 6.8. At this generation, the best point is found to be  $(3.003, 1.994)^T$  with a function value 0.001. The



**Figure 6.8** All 20 points in the population at generation 25 shown on the contour plot of the objective function. The figure shows that most points are clustered around the true minimum.

fitness value at this point is equal to 0.999 and the average population fitness of the population is 0.474. The figure shows how points are clustered around the true minimum of the function in this generation. A few inferior points are still found in the plot. They are the result of some unsuccessful crossover events. We also observe that the total number of function evaluations required to obtain this solution is  $0.8 \times 20 \times 26$  or 416 (including the evaluations of the initial population).

In order to show the efficiency of GAs in arriving at a point close to the true optimum, we perform two more simulations starting with different initial populations. Figure 6.9 shows how the function value of the best point in a population reduces with generation number. Although all three runs have a different initial best point, they quickly converge to a solution close to the true optimum (recall that the optimum point has a function value equal to zero).



**Figure 6.9** The function value of the best point in the population for three independent GA runs. All runs quickly converge to a point near the true optimum.

In order to illustrate the schema processing through genetic operators, we investigate the growth of a particular schema  $H = (0 * \dots * * \dots *)$ . This schema represents all points in the range  $0 \leq x_2 < 3$ . The optimum point lies in this region. With reference to Equation (6.3), we observe that the order, defining length, and the fitness of the schema are such that it is a building block. This schema contains more good points than its competitor  $H^c = (1 * \dots * * \dots *)$  which represents the range  $3 \leq x_2 \leq 6$ . According to Equation (6.3), the schema  $H$  must increase exponentially due to the action of genetic operators. We observe that in the random initial population the schema  $H$  has nine strings and the schema  $H^c$  has 11 strings. At the end of one generation, the population has 14 strings representing the schema  $H$  and only six strings representing the schema  $H^c$ . We may also investigate other interesting regions in the search space and observe their growth in terms of the number of representative points in the population. Other low-order and above-average schemata are also processed similarly and are combined to form

higher-order and good schemata. This processing of several schemata happens in parallel without any extra book-keeping (Goldberg, 1989). Eventually, this processing forms the optimum or a near-optimum point.

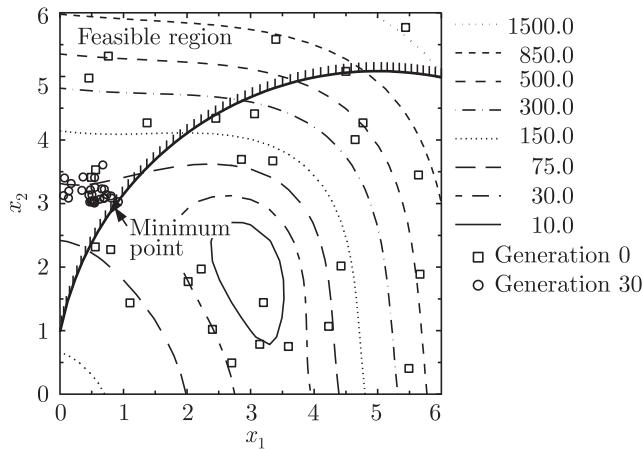
#### 6.1.4 GAs for Constrained Optimization

Genetic algorithms have also been used to solve constrained optimization problems. Although different methods to handle constraints have been suggested, penalty function methods have been mostly used (Deb, 1991; Goldberg, 1983). In the penalty function method, a penalty term corresponding to the constraint violation is added to the objective function. In most cases, a bracket operator penalty term described in Chapter 4 is used. In a constrained minimization problem, the objective function  $f(x)$  is replaced by the penalized function:

$$P(x) = f(x) + \sum_{j=1}^J u_j(g_j(x))^2 + \sum_{k=1}^K v_k[h_k(x)]^2, \quad (6.4)$$

where  $u_j$  and  $v_k$  are penalty coefficients, which are usually kept constant throughout GA simulation. The fitness function is formed by the usual transformation:  $\mathcal{F}(x) = 1/(1+P(x))$ . Since GAs are population based search techniques, the final population converges to a region, rather than a point as depicted in the simulation on Himmelblau's function in Exercise 6.1.1. Unlike the penalty function method described in Chapter 4, the update of penalty parameters in successive sequences is not necessary with GAs. Recall that in the traditional penalty function method (described in Chapter 4), the increase of penalty parameter  $R$  in successive sequences distorted the penalized function. In some occasions, this distortion may create some artificial local optima. This causes the traditional penalty function method difficult to solve a constrained optimization problem in a single sequence. Since GAs can handle multimodal functions better than the traditional methods, a large penalty parameter  $R$  can be used. Since the exact optimum point can only be obtained with an infinite value of penalty parameter  $R$ , in most cases the GA solution would be close to the true optimum. With a solution close to the true optimum, an arbitrary large value of  $R$  can be used with the steepest descent method to find the optimum point in only one sequence of the penalty function method. In order to illustrate this procedure, we reconsider the NLP problem described in Exercise 4.3.1.

With a population of 30 points, a crossover probability of 0.9 and a mutation probability of 0.01, we perform a GA simulation for 30 generations with a penalty parameter  $R = 100$ . This is very large compared to  $R = 0.1$  used in the first sequence in Exercise 4.3.1. Figure 6.10 shows the initial population (with empty boxes) and the population at generation 30 (with empty circles) on the contour plot of the NLP problem. The figure shows that initial population is fairly spread out on the search space. After 30 generations, the complete population is in the feasible region and is placed



**Figure 6.10** Initial population and the population after generation 30 shown on a contour plot of the NLP problem described in Exercise 4.3.1. The hashes used to mark the feasible region is different from that in most other figures in this book.

close to the true optimum point. The best point found at this population is  $(0.809, 3.085)^T$  and may now be used as an initial point for one sequence of the penalty function method described in Chapter 4 (with a large  $R$ ) to obtain the true optimum point with the desired accuracy.

Although the problem contains only two variables and the function is fairly well-behaved, it is interesting to compare the overall function evaluations in GAs and in the penalty function method. Recall that the penalty function method required 609 evaluations in five sequences (the sample run at the end of Chapter 4). In genetic algorithms, one generation required  $(0.9 \times 30)$  or 27 function evaluations. At the end of 30 generations, the total function evaluations required were 837 (including the evaluation of the initial population). It is worth mentioning here that no effort is made to optimally set the GA parameters to obtain the above solution. It is anticipated that with a proper choice of GA parameters, a better result may have been obtained. Nevertheless, for a comparable number of function evaluations, GAs have found a population near the true optimum.

In 2000, the author suggested a penalty-parameter-less constraint-handling approach which has become quite popular in the subsequent years, mainly due to its simplicity and successful performance on most problems (Deb, 2000). The concept is simple. In a tournament selection operator comparing two population members, the following three possibilities and corresponding selection strategy were suggested:

- (i) When one solution is feasible and the other is infeasible, the feasible solution is selected.
- (ii) When both solutions are feasible, the one with better function value is selected.

- (iii) When both solutions are infeasible, the one with smaller *constraint violation* (which we define below) is selected.

The first selection criterion makes sense and is also pragmatic. If a solution cannot be implemented at all, what good is it to know its objective function value? Although there may be some benefits in preserving certain infeasible solutions that are close (in the variable space) to the optimal solution, the above simple strategy seemed to have worked in many difficult problems. The second selection criterion is also obvious and is followed in unconstrained optimization problems. The third criterion makes sense, but a proper definition of a constraint violation may be needed for the procedure to work well. In Deb (2000), we suggested a two-step procedure. First, normalize all constraints using the constant term in the constraint function. For example, the constraint  $g_j(\mathbf{x}) - b_j \geq 0$  is normalized as  $\bar{g}_j(\mathbf{x}) = g_j(\mathbf{x})/b_j - 1 \geq 0$ . Equality constraints can also be normalized similarly to obtain normalized constraint  $\bar{h}_k(\mathbf{x})$ . In constraints where no constant term exists ( $g_j(\mathbf{x}) \geq 0$ ), the constraint can be divided by  $g_j(\bar{\mathbf{x}})$ , where  $\bar{\mathbf{x}} = 0.5(\mathbf{x}^{(L)} + \mathbf{x}^{(U)})$ . Second, the constraint violation ( $CV(\mathbf{x})$ ) can be determined as follows:

$$CV(\mathbf{x}) = \sum_{j=1}^J \langle \bar{g}_j(\mathbf{x}) \rangle + \sum_{k=1}^K |\bar{h}_k(\mathbf{x})|. \quad (6.5)$$

Note that the above function always takes nonnegative values. When the constraint violation is zero, the solution is feasible. Thus, in the early generations while solving complex problems, most or all population members may be infeasible. Then, the above selection operation will face the third scenario most often and will emphasize the solutions that have smaller constraint violations. Thus, early on, a GA with the above approach will attempt to progress towards the feasible region and when a few feasible solutions emerge, they will be emphasized by the first scenario and later on when most solutions are feasible the second scenario will be predominant.

Another way to use the above penalty-parameter-less approach is to convert the problem into the following unconstrained fitness function:

$$\mathcal{F}(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \text{ is feasible}, \\ f_{\max} + CV(\mathbf{x}), & \text{if } \mathbf{x} \text{ is infeasible}, \end{cases} \quad (6.6)$$

where  $f_{\max}$  is the maximum objective function value among the feasible solutions in a population. If a population does not have any feasible solution,  $f_{\max}$  can be chosen as zero. Thus, when no feasible solution exists in the population, the GA minimizes constraint violation and when some feasible solutions exist, the GA emphasizes them more than the infeasible solutions.

Constrained optimization has received a lot of attention in the recent past. Various methods have been tried and the following are worth mentioning (Deb and Datta, 2010; Takahoma and Sakai, 2006; Brest, 2009).

### 6.1.5 Other GA Operators

There exist a number of variations to GA operators. In most cases, the variants are developed to suit particular applications. Nevertheless, there are some variants which are developed in order to achieve some fundamental change in the working of GAs. Here we discuss two such variants for reproduction and crossover operators.

It has been discussed earlier that the roulette-wheel selection operator has inherent noise in selecting good individuals. Although this noise can be somewhat reduced by using stochastic remainder selection, there are two other difficulties with these selection operators. If a population contains an exceptionally good individual early on in the simulation<sup>4</sup>, the expected number of copies in the mating pool ( $\mathcal{F}_i/\bar{\mathcal{F}}$ ) may be so large that the individual occupies most of the mating pool. This reduces the diversity in the mating pool and causes GAs to prematurely converge to a wrong solution. On the other hand, the whole population usually contains equally good points later in the simulation. This may cause each individual to have a copy in the mating pool, thereby making a direction-less search. Both these difficulties can be eliminated by transforming the fitness function  $\mathcal{F}(x)$  to a scaled fitness function  $\mathcal{S}(x)$  at every generation. The transformation could be a simple linear transformation

$$\mathcal{S}(x) = a\mathcal{F}(x) + b.$$

The parameters  $a$  and  $b$  should be defined to allocate the best individual in the population a predefined number of copies in the mating pool and to allocate an average individual one copy in the mating pool (Goldberg, 1989). Since this transformation is performed at every iteration, both difficulties can be eliminated by using this scaling procedure. Another way to overcome the above difficulty is to use a different selection algorithm altogether. *Tournament* selection works by first picking  $s$  individuals (with or without replacement) from the population and then selecting the best of the chosen  $s$  individuals. If performed without replacement in a systematic way<sup>5</sup>, this selection scheme can assign exactly  $s$  copies of the best individual to the mating pool at every generation. The control of this selection *pressure* in tournament selection is making it popular in recent GA applications. In most GA applications, a binary tournament selection with  $s = 2$  is used.

In trying to solve problems with many variables, the single-point crossover operator described earlier may not provide adequate search. Moreover, the single-point crossover operator has some bias of exchange for the right-most

---

<sup>4</sup>This may happen in constrained optimization problems where the population may primarily contain infeasible points except a few feasible points.

<sup>5</sup>First the population is shuffled. Thereafter, the first  $s$  copies are picked from the top of the shuffled list and the best is chosen for the mating pool. Then, the next  $s$  individuals (numbered  $(s+1)$  to  $2s$  in the shuffled list) are picked and the best is chosen for the mating pool. This process is continued until all population members are considered once. The whole population is shuffled again and the same procedure is repeated. This is continued until the complete mating pool is formed.

bits. They have a higher probability of getting exchanged than the left-most bits in the string. Thus, if ten variables are coded left to right with the first variable being at the left-most position and the tenth variable at the right-most position, the effective search on the tenth variable is more compared to the first variable. In order to overcome this difficulty, a multi-point crossover is often used. The operation of a two-point crossover operator is shown below:

$$\begin{array}{c|cc|cc} 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 \end{array} \Rightarrow \begin{array}{c|cc|cc} 0 & 1 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 1 & 1 \end{array}$$

Two random sites are chosen along the string length and bits inside the cross-sites are swapped between the parents. An extreme of the above crossover operator is to have a *uniform* crossover operator where a bit at any location is chosen from either parent with a probability 0.5. In the following, we show the working of a uniform crossover operator, where the first and the fourth bit positions have been exchanged.

$$\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & & 1 & 0 & 0 & 1 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 & & 0 & 1 & 1 & 0 & 1 \end{array} \Rightarrow$$

This operator has the maximum search power among all of the above crossover operators. Simultaneously, this crossover has the minimum survival probability for the good bit combinations (schemata) from parents to children.

#### 6.1.6 Real-coded GAs

As GAs use a coding of variables, they work with a discrete search space. Even though the underlying objective function is a continuous function, GAs convert the search space into a discrete set of points. In order to obtain the optimum point with a desired accuracy, strings of sufficient length need to be chosen. GAs have also been developed to work directly with continuous variables (instead of discrete variables). In those GAs, binary strings are not used. Instead, the variables are directly used. Once a population of random sets of points is created, a reproduction operator (roulette-wheel or other selection operators) can be used to select good strings in the population. In order to create new strings, the crossover and mutation operators described earlier cannot be used efficiently. Even though simple single-point crossover can be used on these points by forcing the cross-sites to fall only on the variable boundaries, the search is not adequate. The search process then mainly depends on the mutation operator. This type of GA has been used in earlier studies (Wright, 1991). Recently, new and efficient crossover operators have been designed so that search along variables is also possible. Let us consider  $x_i^{(j)}$  and  $x_i^{(k)}$  values of design variables  $x_i$  in two parent strings  $j$  and

$k$ . The crossover between these two values may produce the following new value (Radcliffe, 1990):

$$x_i^{\text{new}} = (1 - \lambda)x_i^{(j)} + \lambda x_i^{(k)}, \quad 0 \leq \lambda \leq 1. \quad (6.7)$$

The parameter  $\lambda$  is a random number between zero and one. The above equation calculates a new value bracketing  $x_i^{(j)}$  and  $x_i^{(k)}$ . This calculation is performed for each variable in the string. This crossover has a uniform probability of creating a point inside the region bounded by two parents. An extension to this crossover is also suggested to create points outside the range bounded by the parents. Eshelman and Schaffer (1993) have suggested a blend crossover operator (BLX- $\alpha$ ), in which a new point is created uniformly at random from a larger range extending an amount  $\alpha|x_j^{(i)} - x_j^{(k)}|$  on either side of the region bounded by two parents. The crossover operation depicted in Equation (6.7) can also be used to achieve BLX- $\alpha$  by varying  $\lambda$  in the range  $(-\alpha, 1+\alpha)$ . In a number of test problems, Eshelman and Schaffer have observed that  $\alpha = 0.5$  provides good results. One interesting feature of this type of crossover operator is that the created point depends on the location of both parents. If both parents are close to each other, the new point will also be close to the parents. On the other hand, if parents are far from each other, the search is more like a random search. The random search feature of these crossover operators can be relaxed by using a distribution other than random distribution between parents. A recent study using a polynomial probability distribution with a bias towards near-parent points has been found to perform better than BLX-0.5 in a number of test problems (Deb and Agrawal, 1995). Moreover, the distribution of children points with this crossover resembles that of the single-point, binary crossover operator. More studies in this direction are necessary to investigate the efficacy of real-coded GAs.

IN 1995, Deb and Agrawal (1995) suggested a real-parameter recombination operator that mimics the inherent probability distribution of creating children in a binary single-point crossover applied to a real-parameter space. In the so-called *simulated binary crossover* (SBX) operator, a large probability is assigned to a point close to a parent and a small probability to a point away from a parent. The probability distribution is controlled by a user-defined parameter  $\eta_c$  (called the *distribution index*). SBX is implemented variable-wise, that is, for variable  $x_i$ , two parent values of that variable ( $p_1 = x_i^{(1)}$  and  $p_2 = x_i^{(2)}$ , assuming  $p_2 > p_1$ ) of two solutions  $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$  are recombined (blended) to create two child values ( $c_1$  and  $c_2$ ) as follows:

$$c_1 = \frac{p_1 + p_2}{2} - \beta_L \frac{p_2 - p_1}{2}, \quad (6.8)$$

$$c_2 = \frac{p_1 + p_2}{2} + \beta_R \frac{p_2 - p_1}{2}, \quad (6.9)$$

where the parameters  $\beta_L$  and  $\beta_R$  depend on the random numbers  $u_L$  and  $u_R$  (both lie within  $[0, 1]$ ) and are defined as follows:

$$\beta_L = \begin{cases} (2u_L(1 - \alpha_L))^{1/(\eta_c+1)}, & \text{for } 0 \leq u_L \leq 0.5/(1 - \alpha_L), \\ \frac{1}{2(1 - u_L(1 - \alpha_L))^{1/(\eta_c+1)}}, & \text{for } 0.5/(1 - \alpha_L) < u_L \leq 1 \end{cases} \quad (6.10)$$

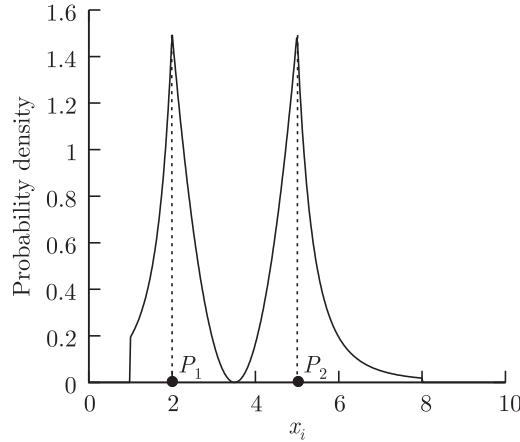
$$\beta_R = \begin{cases} (2u_R(1 - \alpha_R))^{1/(\eta_c+1)}, & \text{for } 0 \leq u_R \leq 0.5/(1 - \alpha_R), \\ \frac{1}{2(1 - u_R(1 - \alpha_R))^{1/(\eta_c+1)}}, & \text{for } 0.5/(1 - \alpha_R) < u_R \leq 1 \end{cases} \quad (6.11)$$

where

$$\alpha_L = \frac{0.5}{\left(1 + 2(p_1 - x_i^{(L)})/(p_2 - p_1)\right)^{\eta_c+1}}, \quad (6.12)$$

$$\alpha_R = \frac{0.5}{\left(1 + 2(x_i^{(U)} - p_2)/(p_2 - p_1)\right)^{\eta_c+1}}. \quad (6.13)$$

The above calculations ensure that created child solutions  $c_1$  and  $c_2$  always lie within the specified variable bounds  $(x_i^{(L)}, x_i^{(U)})$ . For single-objective optimization problems, experimental results have shown that  $\eta_c \approx 2$  produces good results (Deb and Agrawal, 1995). The larger the value of  $\eta_c$ , the smaller is the difference between parents and respective created child. Figure 6.11 shows the probability distribution of children solutions near two parents ( $p_1 = 2.0$  and  $p_2 = 5.0$ ) with  $\eta_c = 2$  and having bounds  $[1, 8]$ . Notice that



**Figure 6.11** Probability density function of creating a child solution from two parents  $p_1 = 2.0$  and  $p_2 = 5.0$  over the entire search space  $[1, 8]$  is shown.

the probability density is zero outside the specified variable bounds. It is also clear that points near  $p_1 = 2.0$  and  $p_2 = 5.0$  are more likely to be created than away from them.

A similar mutation operator can also be devised for real-parameter GAs. Here, we describe the commonly used *polynomial mutation* operator (Deb, 2001). As the name suggests, a polynomial probability distribution is used to create a mutated solution. The order of the polynomial ( $\eta_m$ ) is a user-defined parameter. This operator is also applied variable-wise. For a given parent solution  $p \in [x_i^{(L)}, x_i^{(U)}]$ , the mutated solution  $p'$  for a particular variable is created for a random number  $u$  created within  $[0,1]$ , as follows:

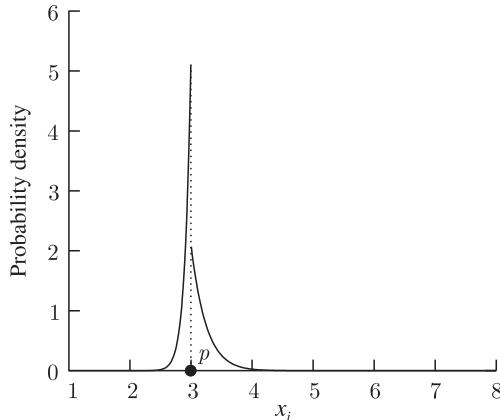
$$p' = \begin{cases} p + \bar{\delta}_L(p - x_i^{(L)}), & \text{for } u \leq 0.5, \\ p + \bar{\delta}_R(x_i^{(U)} - p), & \text{for } u > 0.5. \end{cases} \quad (6.14)$$

Then, either of the two parameters ( $\bar{\delta}_L$  or  $\bar{\delta}_R$ ) is calculated, as follows:

$$\bar{\delta}_L = (2u)^{1/(1+\eta_m)} - 1, \quad \text{for } u \leq 0.5, \quad (6.15)$$

$$\bar{\delta}_R = 1 - (2(1-u))^{1/(1+\eta_m)}, \quad \text{for } u > 0.5. \quad (6.16)$$

The parameter  $\eta_m$  is usually chosen in the range  $[20, 100]$ . The larger the value, the smaller is the difference between the parent and the created child solution. The above calculations ensure that the created mutated child is always bounded within the specified lower and upper bounds. Figure 6.12 shows the probability density of creating a mutated child point from a parent point  $p = 3$  in a bounded range of  $[1, 8]$  with  $\eta_m = 20$ . Notice how with  $\eta_m = 20$  points only near the parent ( $x_i = 3$ ) can be created. Also since the



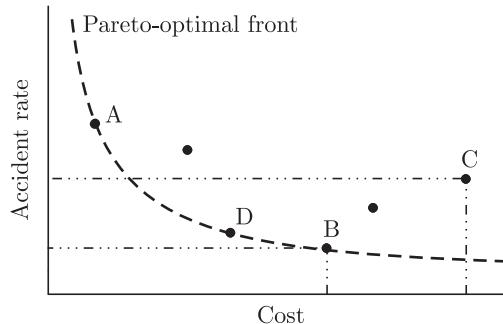
**Figure 6.12** Probability density function of creating a mutated child solution from a parent  $p = 3.0$  over the entire search space  $[1, 8]$  is shown.

lower boundary is closer to the parent solution, points are created more closer to the parent in the left side than that in the right side, although the overall probability of creating a mutated child is equal in left and right sides of the

parent. Thus, when a solution lying on a boundary (say, on the left bound) is mutated, there is 50% probability for the point to remain on the same boundary and the remaining 50% probability exists for an inside solution to become a mutated child.

### 6.1.7 Multi-objective GAs

Many engineering design problems involve simultaneous solution of multiple objective functions. The most common problem which arises in engineering design is to minimize the overall cost of manufacturing and simultaneously minimize the associated accident rate (failures or rejections). Consider Figure 6.13, where a number of plausible solutions to a hypothetical design are shown. Solution A costs less but is more accident-prone.



**Figure 6.13** A typical two-objective problem. Solutions A, B and D are Pareto-optimal solutions.

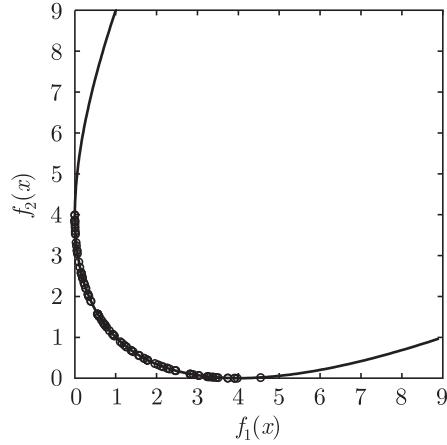
Solution B, on the other hand, costs more, but is less prone to accidents. In their respective ways, both these solutions are useful, but solution C is not as good as solution B in both objectives. It incurs more cost as well as more accidents. Thus, in multi-objective optimization problems, there exist a number of solutions which are optimum in some sense. These solutions constitute a Pareto-optimal front shown by the thick-dashed line in the figure. Since any point on this front is an optimum point, it is desirable to find as many such points as possible. Recently, a number of extensions to simple GAs have been tried to find many of these Pareto-optimal points simultaneously (Horn and Nafpliotis, 1993; Fonseca and Fleming, 1993; Srinivas and Deb, 1995). Consider the following two objective functions for minimization:

$$f_1(x) = x^2,$$

$$f_2(x) = (x - 2)^2,$$

in the interval  $-1000 \leq x \leq 1000$ . The Pareto-optimal front for this problem is the complete region  $0 \leq x \leq 2$ . With a random initial population of 100 points in the range  $-1000 \leq x \leq 1000$ , the modified GA converges to the range

$0 \leq x \leq 2$  with a population distribution shown in Figure 6.14 (Srinivas and Deb, 1995). The modified GA has also been successfully applied to other test functions and a truss-structure optimal design problem (Srinivas, 1994).



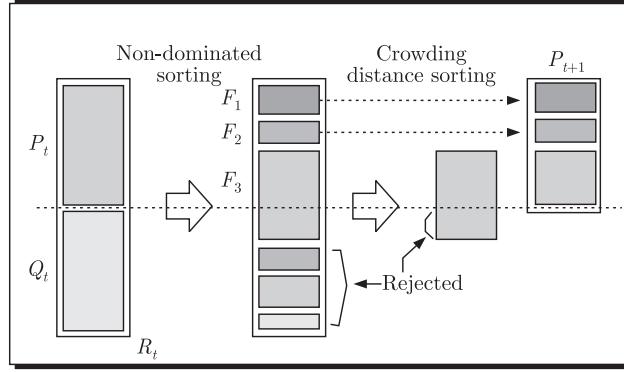
**Figure 6.14** Population of 100 points at generation 100.

The NSGA algorithm was modified by the author and his students to devise an elitist NSGA or NSGA-II which has received a lot of attention since its original suggestion in 2000 (Deb et al., 2000) and then a modified version in 2002 (Deb et al., 2002). The NSGA-II algorithm has the following three features:

- (i) It uses an elitist principle,
- (ii) It uses an explicit diversity preserving mechanism.
- (iii) It emphasizes non-dominated solutions.

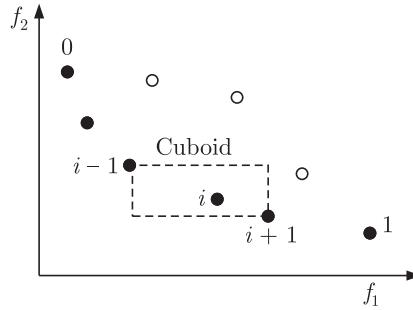
At any generation  $t$ , the offspring population (say,  $Q_t$ ) is first created by using the parent population (say,  $P_t$ ) and the usual genetic operators. Thereafter, the two populations are combined together to form a new population (say,  $R_t$ ) of size  $2N$ . Then, the population  $R_t$  is classified into different non-dominated classes. Thereafter, the new population is filled by points of different non-dominated fronts, one at a time. The filling starts with the first non-dominated front (of class one) and continues with points of the second non-dominated front, and so on. Since the overall population size of  $R_t$  is  $2N$ , not all fronts can be accommodated in  $N$  slots available for the new population. All fronts which could not be accommodated are deleted. When the last allowed front is being considered, there may exist more points in the front than the remaining slots in the new population. This scenario is illustrated in Figure 6.15. Instead of arbitrarily discarding some members from the last front, the points which will make the diversity of the selected points the highest are chosen.

The crowded-sorting of the points of the last front which could not be accommodated fully is achieved in the descending order of their *crowding*



**Figure 6.15** Schematic of the NSGA-II procedure.

*distance* values and points from the top of the ordered list are chosen. The crowding distance  $d_i$  of point  $i$  is a measure of the objective space around  $i$  which is not occupied by any other solution in the population. Here, we simply calculate this quantity  $d_i$  by estimating the perimeter of the cuboid (Figure 6.16) formed by using the nearest neighbours in the objective space as the vertices (we call this the *crowding distance*).



**Figure 6.16** The crowding distance calculation.

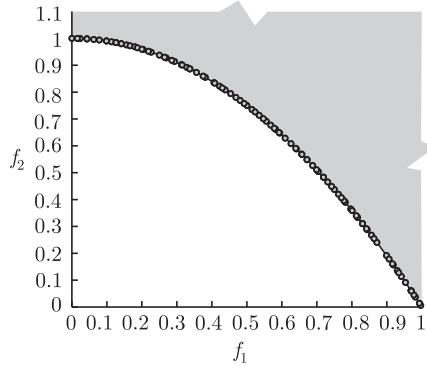
Here, we show results from several runs of the NSGA-II algorithm on two test problems. The first problem (ZDT2) is two-objective, 30-variable problem with a concave Pareto-optimal front:

$$\text{ZDT2 : } \left\{ \begin{array}{l} \text{Minimize } f_1(\mathbf{x}) = x_1, \\ \text{Minimize } f_2(\mathbf{x}) = s(\mathbf{x}) [1 - (f_1(\mathbf{x})/s(\mathbf{x}))^2], \\ \text{where } s(\mathbf{x}) = 1 + \frac{9}{29} \sum_{i=2}^{30} x_i \\ 0 \leq x_1 \leq 1, \\ -1 \leq x_i \leq 1, \quad i = 2, 3, \dots, 30. \end{array} \right. \quad (6.17)$$

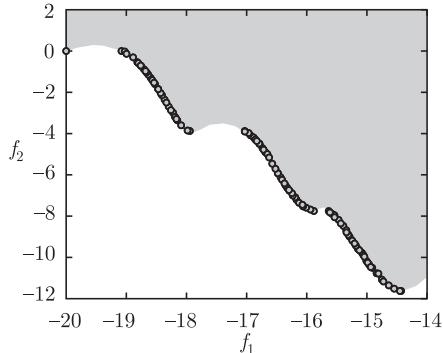
The second problem (KUR), with three variables, has a disconnected Pareto-optimal front:

$$\text{KUR : } \begin{cases} \text{Minimize} & f_1(\mathbf{x}) = \sum_{i=1}^2 \left[ -10 \exp(-0.2\sqrt{x_i^2 + x_{i+1}^2}) \right], \\ \text{Minimize} & f_2(\mathbf{x}) = \sum_{i=1}^3 [|x_i|^{0.8} + 5 \sin(x_i^3)] , \\ & -5 \leq x_i \leq 5, \quad i = 1, 2, 3. \end{cases} \quad (6.18)$$

NSGA-II is run with a population size of 100 and for 250 generations. The variables are used as real numbers and an SBX recombination operator (Deb and Agrawal, 1995) with  $p_c = 0.9$  and distribution index of  $\eta_c = 10$  and a polynomial mutation operator (Deb, 2001) with  $p_m = 1/n$  ( $n$  is the number of variables) and distribution index of  $\eta_m = 20$  are used. Figures 6.17 and 6.18 show that NSGA-II converges to the Pareto-optimal front and maintains a good spread of solutions on both test problems.



**Figure 6.17** NSGA-II on ZDT2.



**Figure 6.18** NSGA-II on KUR.

There also exist other competent EMOs, such as strength Pareto evolutionary algorithm (SPEA) and its improved version SPEA2

(Zitzler et al., 2001), Pareto archived evolution strategy (PAES) and its improved versions PESA and PESA2 (Corne et al., 2000), multi-objective messy GA (MOMGA) (Veldhuizen and Lamont, 2000), multi-objective Micro-GA (Coello and Toscano, 2000), neighbourhood constraint GA (Loughlin and Ranjithan, 1997), ARMOGA (Sasaki et al., 2001) and others. Besides, there exists other EA-based methodologies, such as Particle swarm EMO (Coello and Lechuga, 2002; Mostaghim and Teich, 2003), Ant-based EMO (McMullen, 2001; Gravel et al., 2002), and differential evolution-based EMO (Babu and Jehan, 2003).

NSGA-II employs an efficient constraint-handling procedure. The constraint-handling method modifies the binary tournament selection, where two solutions are picked from the population and the better solution is chosen. In the presence of constraints, each solution can be either feasible or infeasible. Thus, there may be at most three situations: (i) both solutions are feasible, (ii) one is feasible and the other is not, and (iii) both are infeasible. We consider each case by simply redefining the domination principle as follows (we call it the *constrained-domination* condition for any two solutions  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(j)}$ ): A solution  $\mathbf{x}^{(i)}$  is said to ‘constrained-dominate’ a solution  $\mathbf{x}^{(j)}$  (or  $\mathbf{x}^{(i)} \preceq_c \mathbf{x}^{(j)}$ ), if any of the following conditions are true:

- (i) Solution  $\mathbf{x}^{(i)}$  is feasible and solution  $\mathbf{x}^{(j)}$  is not.
- (ii) Solutions  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(j)}$  are both infeasible, but solution  $\mathbf{x}^{(i)}$  has a smaller constraint violation, which can be computed by adding the normalized violation of all constraints:

$$\text{CV}(\mathbf{x}) = \sum_{j=1}^J \max(0, -\bar{g}_j(\mathbf{x})) + \sum_{k=1}^K \text{abs}(\bar{h}_k(\mathbf{x})).$$

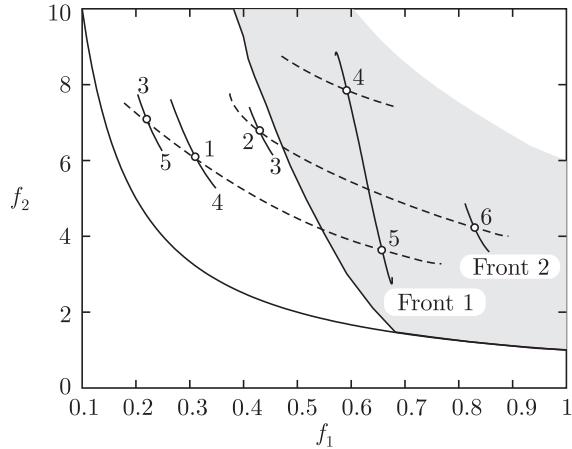
The normalization of a constraint  $g_j(\mathbf{x}) \geq g_{j,r}$  can be achieved as  $\bar{g}_j(\mathbf{x}) \geq 0$ , where  $\bar{g}_j(\mathbf{x}) = g_j(\mathbf{x})/g_{j,r} - 1$ .

- (iii) Solutions  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(j)}$  are feasible and solution  $\mathbf{x}^{(i)}$  dominates solution  $\mathbf{x}^{(j)}$  in the usual sense.

The above change in the definition requires a minimal change in the NSGA-II procedure described earlier. Figure 6.19 shows the non-dominated fronts on a six-member population due to the introduction of two constraints (the minimization problem is described as CONSTR elsewhere (Deb, 2001)). In the absence of the constraints, the non-dominated fronts (shown by dashed lines) would have been ((1,3,5), (2,6), (4)), but in their presence, the new fronts are ((4,5), (6), (2), (1), (3)). The first non-dominated front consists of the “best” (that is, non-dominated and feasible) points from the population and any feasible point lies on a better non-dominated front than an infeasible point.

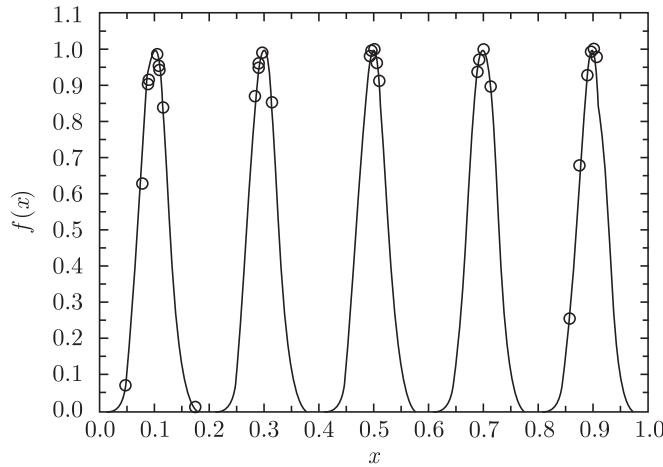
### 6.1.8 Other Advanced GAs

One advantage with a population-based search technique is that a number of solutions can be captured in the population. GAs have exploited this feature



**Figure 6.19** Non-constrained-domination fronts are shown.

to find multiple optimal solutions simultaneously in multimodal function optimization problems (Deb, 1989; Goldberg et al., 1992; Goldberg and Richardson, 1987). Consider the five-peaked function shown in Figure 6.20. Since all five maximum points have equal function values, designers may be interested in finding all five optimum points (or as many as possible). The knowledge of multiple optimal points enables designers to adapt to different optimal solutions, as and when required. The population in a GA simulation is adaptively divided into separate subpopulations corresponding to each optimum point by using *sharing functions* (Goldberg and Richardson, 1987). GAs with an initial population of 50 random points have converged to a population shown in Figure 6.20 after 100 generations, finding all five optimum points simultaneously. Any further discussion on the technique of multimodal



**Figure 6.20** A function with five maximum points. A population of 50 points after 100 generations of a GA simulation with sharing functions shows that all optimum points are found.

optimization using GAs is beyond the scope of this book. Interested readers may refer to the above-mentioned GA literature for more details.

Besides, a plethora of other applications including non-stationary function optimization, job shop scheduling problems, routing problems, and travelling salesperson problems have been tried using GAs. Among engineering applications, structural optimization problems (Hajela, 1990; Jenkins, 1991; Deb, 1991; Rajeev and Krishnamoorthy, 1992), turbine blade design problem (Powell et al., 1989), laminate stacking problem in composite structures (Callahan and Weeks, 1992), and pipeline optimization (Goldberg, 1983) are some of the applications. The growing list of successful applications of GAs to different engineering problems confirms the robustness of GAs and shows promise of GAs in solving other engineering design problems.

## 6.2 Simulated Annealing

The simulated annealing method resembles the cooling process of molten metals through annealing. At high temperature, the atoms in the molten metal can move freely with respect to each other, but as the temperature is reduced, the movement of the atoms gets restricted. The atoms start to get ordered and finally form crystals having the minimum possible energy. However, the formation of the crystal mostly depends on the cooling rate. If the temperature is reduced at a very fast rate, the crystalline state may not be achieved at all, instead, the system may end up in a polycrystalline state, which may have a higher energy state than the crystalline state. Therefore, in order to achieve the absolute minimum energy state, the temperature needs to be reduced at a slow rate. The process of slow cooling is known as *annealing* in metallurgical parlance.

The simulated annealing procedure simulates this process of slow cooling of molten metal to achieve the minimum function value in a minimization problem. The cooling phenomenon is simulated by controlling a temperature-like parameter introduced with the concept of the Boltzmann probability distribution. According to the Boltzmann probability distribution, a system in thermal equilibrium at a temperature  $T$  has its energy distributed probabilistically according to  $P(E) = \exp(-E/kT)$ , where  $k$  is the Boltzmann constant. This expression suggests that a system at a high temperature has almost uniform probability of being at any energy state, but at a low temperature it has a small probability of being at a high energy state. Therefore, by controlling the temperature  $T$  and assuming that the search process follows the Boltzmann probability distribution, the convergence of an algorithm can be controlled. Metropolis, et al. (1953) suggested one way to implement the Boltzmann probability distribution in simulated thermodynamic systems. The same can also be used in the function minimization context. Let us say, at any instant the current point is  $x^{(t)}$  and the function value at that point is  $E(t) = f(x^{(t)})$ . Using the Metropolis algorithm, we can say that the probability of the next point being at  $x^{(t+1)}$

depends on the difference in the function values at these two points or on  $\Delta E = E(t+1) - E(t)$  and is calculated using the Boltzmann probability distribution:

$$P(E(t+1)) = \min [1, \exp(-\Delta E/kT)].$$

If  $\Delta E \leq 0$ , this probability is one and the point  $x^{(t+1)}$  is always accepted. In the function minimization context, this makes sense because if the function value at  $x^{(t+1)}$  is better than that at  $x^{(t)}$ , the point  $x^{(t+1)}$  must be accepted. The interesting situation happens when  $\Delta E > 0$ , which implies that the function value at  $x^{(t+1)}$  is worse than that at  $x^{(t)}$ . According to many traditional algorithms, the point  $x^{(t+1)}$  must not be chosen in this situation. But according to the Metropolis algorithm, there is some finite probability of selecting the point  $x^{(t+1)}$  even though it is a worse than the point  $x^{(t)}$ . However, this probability is not the same in all situations. This probability depends on relative magnitude of  $\Delta E$  and  $T$  values. If the parameter  $T$  is large, this probability is more or less high for points with largely disparate function values. Thus, any point is almost acceptable for a large value of  $T$ . On the other hand, if the parameter  $T$  is small, the probability of accepting an arbitrary point is small. Thus, for small values of  $T$ , the points with only small deviation in function value are accepted. Later, we shall illustrate this feature of simulated annealing algorithm better through an exercise problem.

Simulated annealing is a point-by-point method. The algorithm begins with an initial point and a high temperature  $T$ . A second point is created at random in the vicinity of the initial point and the difference in the function values ( $\Delta E$ ) at these two points is calculated. If the second point has a smaller function value, the point is accepted; otherwise the point is accepted with a probability  $\exp(-\Delta E/T)$ . This completes one iteration of the simulated annealing procedure. In the next generation, another point is created at random in the neighbourhood of the current point and the Metropolis algorithm is used to accept or reject the point. In order to simulate the thermal equilibrium at every temperature, a number of points ( $n$ ) is usually tested at a particular temperature, before reducing the temperature. The algorithm is terminated when a sufficiently small temperature is obtained or a small enough change in function values is found.

### Algorithm

**Step 1** Choose an initial point  $x^{(0)}$ , a termination criterion  $\epsilon$ . Set  $T$  a sufficiently high value, number of iterations to be performed at a particular temperature  $n$ , and set  $t = 0$ .

**Step 2** Calculate a neighbouring point  $x^{(t+1)} = N(x^{(t)})$ . Usually, a random point in the neighbourhood is created.

**Step 3** If  $\Delta E = E(x^{(t+1)}) - E(x^{(t)}) < 0$ , set  $t = t + 1$ ;  
Else create a random number ( $r$ ) in the range  $(0, 1)$ . If  $r \leq \exp(-\Delta E/T)$  set  $t = t + 1$ ;  
Else go to Step 2.

**Step 4** If  $|x^{(t+1)} - x^{(t)}| < \epsilon$  and  $T$  is small, **Terminate**;  
 Else if  $(t \bmod n) = 0$  then lower  $T$  according to a cooling schedule. Go to Step 2;  
 Else go to Step 2.

The initial temperature ( $T$ ) and the number of iterations ( $n$ ) performed at a particular temperature are two important parameters which govern the successful working of the simulated annealing procedure. If a large initial  $T$  is chosen, it takes a number of iterations for convergence. On the other hand, if a small initial  $T$  is chosen, the search is not adequate to thoroughly investigate the search space before converging to the true optimum. A large value of  $n$  is recommended in order to achieve quasi-equilibrium state at each temperature, but the computation time is more. Unfortunately, there are no unique values of the initial temperature and  $n$  that work for every problem. However, an estimate of the initial temperature can be obtained by calculating the average of the function values at a number of random points in the search space. A suitable value of  $n$  can be chosen (usually between 20 and 100) depending on the available computing resource and the solution time. Nevertheless, the choice of the initial temperature and subsequent cooling schedule still remain an art and usually require some trial-and-error efforts. We illustrate the working of this algorithm on the unconstrained Himmelblau's function.

### EXERCISE 6.2.1

The objective of this exercise is to minimize the following problem using the simulated annealing method.

$$\text{Minimize } f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

**Step 1** We choose an initial point  $x^{(0)} = (2.5, 2.5)^T$ , and a termination factor  $\epsilon = 10^{-3}$ . In order to choose an initial temperature, we compute the average of the function values at four points  $(0, 0)^T$ ,  $(0, 5)^T$ ,  $(5, 0)^T$ , and  $(5, 5)^T$ . We obtain the initial temperature  $T = 405.0$ . For brevity, we reduce the number of iteration steps to  $n = 1$ . We also set the initial iteration counter  $t = 0$ .

**Step 2** At this step, we first create a point in the neighbourhood of the point  $x^{(0)}$ . In order to achieve that, we choose two numbers (one for each variable) according to the Gaussian distribution. The mean of the distribution is kept at the current point and the variance is assigned so as to have about 99.73 per cent (equivalent to  $3\sigma$ ) of the points in the given domain ( $0 \leq x_1, x_2 \leq 5$ ). This variance is found by assuming that the mean is at the mid-point of the range. Thus, we solve the equation:  $3\sigma = (5 - 0)/2$  and obtain  $\sigma = 0.833$ . With this distribution, we obtain two numbers  $\Delta x_1 = 0.037$  and  $\Delta x_2 = -0.086$ , respectively. Thus, the new point is  $x^{(1)} = (2.537, 2.414)^T$  with a function value  $f(x^{(1)}) = 6.482$ . The initial point had a function value equal to  $f(x^{(0)}) = 8.125$ .

**Step 3** Thus, the change in the function value is equal to  $\Delta E = f(x^{(1)}) - f(x^{(0)}) = -1.643$ . Since  $\Delta E < 0$ , we accept the new point. We increment the counter  $t = 1$  and proceed to Step 4.

**Step 4** Since the two points  $(x^{(0)}$  and  $x^{(1)}$ ) are not close enough for termination, we do not terminate. This completes one iteration of the simulated annealing algorithm. In practice, a number of such iterations (20 to 100) is tried before reducing the temperature. But here we reduce the temperature after every iteration to show the working of the algorithm in a limited number of iterations. We also choose to decrement the temperature parameter  $T$  by a factor 0.5 at every iteration. This factor is usually set to a larger value, in practice. At the end of this step, the new temperature is  $T = 0.5 \times 405 = 202.5$ .

**Step 2** At this step, we create another new point in the neighbourhood of the point  $x^{(1)}$ . Following the procedure as in the previous iteration, we create two numbers based on the Gaussian distribution with zero mean:  $\Delta x_1 = -0.426$  and  $\Delta x_2 = -1.810$ . The corresponding point is  $x^{(2)} = (2.072, 0.604)^T$ .

**Step 3** The difference in the function values is

$$\Delta E = f(x^{(2)}) - f(x^{(1)}) = 58.067 - 6.482 = 51.585.$$

Since this quantity is positive, we use Metropolis algorithm to decide whether to accept or reject the point. We choose a random number in the range  $(0, 1)$ :  $r = 0.649$ . The probability for accepting the new point is  $\exp(-51.585/202.5) = 0.775$ . Since  $r < 0.775$ , we accept this point. It is interesting to note that even though the function value at the point  $x^{(2)}$  is almost nine times worse than that at the point  $x^{(1)}$ , the point  $x^{(2)}$  has been accepted. Since, this is an early stage of the simulation, the temperature is high and a large perturbation is allowed. This property of the algorithm does not allow it to prematurely converge to a wrong solution. We set  $t = 2$  and proceed to Step 4.

**Step 4** The termination criterion is not met here too. Therefore, we decrease the temperature to  $T = 0.5 \times 202.5 = 101.25$ .

**Step 2** The next point found in the vicinity of the current point is  $x^{(3)} = (2.397, -0.312)^T$ . This point has a function value equal to  $f(x^{(3)}) = 51.287$ .

**Step 3** Since the difference in the function values  $\Delta E = -6.780$  is negative, we accept the point. Thus,  $t = 3$  and we move to Step 4.

**Step 4** We decrease the temperature to  $T = 50.625$  and proceed to Step 2.

**Step 2** The point  $x^{(4)} = (1.397, 1.721)^T$  is found in the vicinity of the point  $x^{(3)}$ . The function value at the new point is  $f(x^{(4)}) = 60.666$ . Since the difference in the function values is  $\Delta E = f(x^{(4)}) - f(x^{(3)}) = 9.379$ , we use the Metropolis algorithm. We create a random number in the

range  $(0, 1)$ :  $r = 0.746$ . The probability of accepting the new point is  $\exp(-9.379/50.625) = 0.831$ . Since,  $r < 0.831$ , we accept this point also. Even though the temperature is comparatively low, this point is accepted because the difference in the objective function value is small. We increment the iteration counter  $t = 4$  and move to Step 4.

**Step 4** We decrease the temperature to  $T = 25.313$  and proceed to Step 2.

**Step 2** The point created at this step is  $x^{(5)} = (0.793, 1.950)^T$  with a function value equal to  $f(x^{(5)}) = 76.697$ .

**Step 3** Since the difference in function values at  $x^{(5)}$  and  $x^{(4)}$  is  $\Delta E = 16.031$ , which is positive, we use the Metropolis algorithm again. We create a random number  $r = 0.793$ . The probability of accepting the point is  $\exp(-16.031/25.313) = 0.531$ . Since  $r > 0.531$ , we do not accept the point. This situation arises when the temperature is comparatively smaller than the difference in function value. Thus, the temperature acts as a measure of tolerance allowed between two successive function values for accepting a worse point. We proceed to Step 2 to create a new point.

**Step 2** Another point in the vicinity of the point  $x^{(5)}$  is found:  $x^{(6)} = (1.691, 2.089)^T$ . The function value at this point is  $f(x^{(6)}) = 37.514$ .

**Step 3** Since the difference in function values at  $x^{(6)}$  and  $x^{(5)}$  ( $\Delta E = -23.152$ ) is negative, the point  $x^{(6)}$  is better than the earlier point  $x^{(5)}$ . Thus, we accept this point.

**Step 4** The temperature at this step is  $T = 12.656$ .

This process continues until the temperature is reduced to a small value. With proper choice of parameters, the simulated annealing algorithm can find the global optimal point. In early stages of the simulation, any point is equally likely to be accepted. Thus, the search space is well investigated before converging to any solution. This increases the probability of finding the global optimal point. In later stages, the temperature is small and the search process allows only minor changes in the solution, thereby achieving a solution with the desired accuracy. In fact, for sufficiently large number of iterations at each temperature and with a small cooling rate, the algorithm guarantees convergence to the globally optimal solution (Aarts and Korst, 1989). The simulated annealing procedure can also be used to solve constrained optimization problems by using the exterior penalty function method (Kirpatrick et al., 1983), similar to the method described in Section 6.1.4. The energy function  $E$  can be replaced with the penalized function value. In the following section, we compare the power of these nontraditional algorithms with a traditional search algorithm in finding the global optimum point in a multimodal function.

### 6.3 Global Optimization

In this section, one traditional optimization algorithm (the steepest descent method) and two nontraditional optimization algorithms described in the previous sections are used to solve a two-dimensional function optimization problem to global optimality. In real-world problems, the objective function usually contains a number of optima of which one (or more) is the global optimum. Other optima have worse function values compared to the one at the global optimum. Therefore, as a designer or a decision-maker, one may be interested in finding the globally optimum point which corresponds to the best function value. Here, first we design a function having four minima of which only one is the global minimum. Thereafter, we apply the steepest descent algorithm, genetic algorithms, and simulated annealing algorithm to compare their performance in finding the global minimum point.

In Chapters 3 and 4, we have chosen the following Himmelblau function:

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

In most applications, only positive values of variables  $x_1$  and  $x_2$  were considered. Thus, the optimum point  $(3, 2)^T$  was found. In fact, the above function has three more minimum points, one at each quadrant. These minimum points can be obtained by solving the following equations:

$$\begin{aligned} x_1^2 + x_2 - 11 &= 0, \\ x_1 + x_2^2 - 7 &= 0. \end{aligned}$$

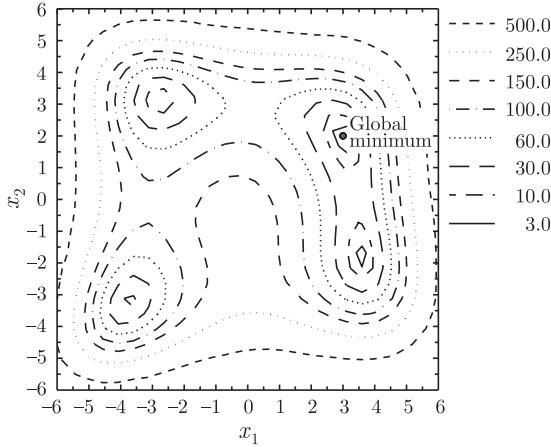
The solutions to these equations are  $(3, 2)^T$ ,  $(-2.805, 3.131)^T$ ,  $(-3.779, -3.283)^T$ , and  $(3.584, -1.848)^T$ . Since all four points satisfy the above two equations, the function values at these points are zero. Thus, the Himmelblau function has four minimum points all having a function value equal to zero. Since we are interested in designing a function with only one global minimum point, we add a term with the Himmelblau function and form the following two-dimensional function:

$$\begin{aligned} \text{Minimize } f(x_1, x_2) &= (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \\ &\quad + 0.1[(x_1 - 3)^2 + (x_2 - 2)^2], \end{aligned} \quad (6.19)$$

where  $-6 \leq x_1, x_2 \leq 6$ . The addition of the term does not alter the location and the function value at the minimum point  $(3, 2)^T$ , but alters locations and function values of the other three minimum points, thereby making them local minimum points. Thus, the global minimum of the above function is at  $(3, 2)^T$  and has a function value equal to zero. Other minima (local minima) have higher function values (3.498, 7.386, and 1.515, respectively). Figure 6.21 shows all minima including the global minimum point.

#### 6.3.1 Using the Steepest Descent Method

The steepest descent algorithm is outlined in Chapter 3. The algorithm is used to solve the problem in Equation (6.19) for global optimality. Recall that the

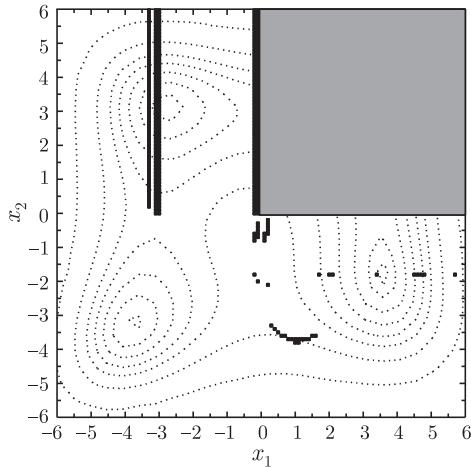


**Figure 6.21** The contour plot of the multimodal objective function. The figure shows the four minima at different locations. The global minimum is also marked.

steepest descent method works from a chosen initial point by finding a search direction along the negative of the gradient at the point. At every iteration, the best point along the search direction is found and the search continues from this point by finding a new search direction along the negative of the gradient at that point. We have highlighted in Chapter 3 that the solution of the steepest descent method largely depends on the chosen initial point. In order to eliminate this bias, we consider 121 points for each variable (uniformly spaced in the range  $x_1, x_2 \in (-6, 6)$ ), thereby making a total of  $121 \times 121$  or 14,641 initial points. Thereafter, we apply the steepest descent algorithm 14,641 times, every time starting from a different point. Since there are many results to report, we have plotted the outcome of all runs in Figure 6.22 in a simple way. If a simulation starting from an initial solution  $x_0$  converges<sup>6</sup> to the vicinity of the global minimum  $(3, 2)^T$ , we mark that initial point  $x_0$  by a small solid box in the figure. This means that all simulation runs starting with points marked in solid boxes in the figure have successfully found the global minimum, whereas rest of the runs could not find the global minimum. It is clear from the plot that the convergence of steepest descent algorithm largely depends on the chosen initial point in the case of problems with multiple optima. Out of 14,641 points, 4,012 points have successfully converged to the global minimum. The successful runs have taken on an average 215 function evaluations for convergence. Since the set of successful initial points contains about one-fourth of the search space (Figure 6.22), it can be concluded that on an average one out of four simulations of the steepest descent algorithm solves the above function to global optimality. This result is typical for many

---

<sup>6</sup>A simulation run is considered to have converged if in two successive sequences, points with three decimal places accuracy are found.



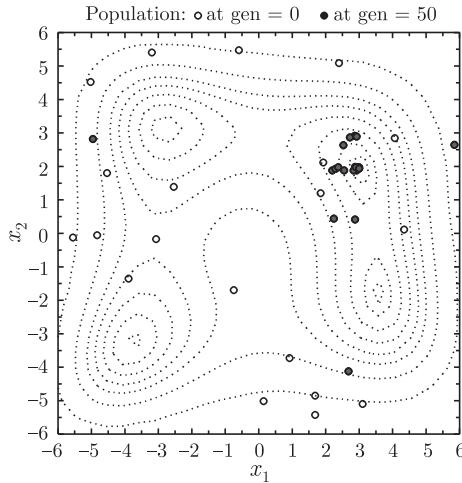
**Figure 6.22** The starting points that successfully converged to the global minimum  $(3, 2)^T$  with the steepest descent method are marked in solid boxes. Approximately 72 per cent of the points could not converge to the global minimum.

traditional optimization algorithms. If they do not begin with a sufficiently good point, the algorithms may converge to a wrong solution.

### 6.3.2 Using Genetic Algorithms

Genetic algorithms are applied next. Recall that in order to use GAs, the variables need to be coded first. A 10-bit binary coding for each variable is chosen. With 10 bits for each variable in the interval  $(-6, 6)^T$ , the obtainable accuracy is 0.012. The total length of a string becomes 20. A population size of  $n = 20$  is used. Binary tournament selection procedure is used. For crossover, the single-point crossover with a crossover probability  $p_c = 0.8$  is used. For mutation, the bit-wise mutation with a mutation probability  $p_m = 0.05$  is chosen. In Figure 6.23, the initial population of points is marked with empty circles. The figure shows that all quadrants have almost equal number of points to start with. The population after 50 generations is also shown in Figure 6.23 (marked with solid circles). The figure shows that the population has converged near the global minimum, even though it could have converged to any of the other three minima. The population at generation 50 contains a point  $(2.997, 1.977)^T$  with a function value equal to 0.0106.

The above result is obtained by simulating GAs on only one initial population. In order to investigate the robustness of GAs, we apply them on a number of different initial populations. As mentioned earlier, the successful working of GAs depends on the proper selection of GA parameters. Thus, we investigate the effect of GA parameters on the number of function evaluations required to solve the above multimodal problem. Table 6.3 shows the outcome of these experiments. For each parameter setting (crossover probability  $p_c$ ,



**Figure 6.23** All 20 points in the initial population and the population at generation 50 shown on the contour plot of the multimodal function. The empty circles represent the initial population and the solid circles represent the population at generation 50.

mutation probability  $p_m$  and population size  $n$ ), 100 different simulations, each starting with a different initial population, are performed. The algorithm is terminated if the best point at any generation comes to within 1 per cent of the global minimum. Since the problem has two variables, a random search technique would take on an average about  $1/(0.01)^2$  or 10,000 function evaluations to achieve a solution within 1 per cent of the global minimum point. Thus, the maximum allowable generation permitted at each GA simulation is chosen in such a way that the maximum number of function evaluations does not exceed 10,000. For example, with a population size of 20 and a crossover probability of 0.8, each generation requires exactly  $(0.8 \times 20)$  or 16 function evaluations. Thus, the maximum allowable generation in this case is  $10,000/16$  or 625.

Table 6.3 shows the number of times (out of 100 runs) that GAs have been able to converge to a solution close to the global minimum and the average number of function evaluations required to achieve that solution.

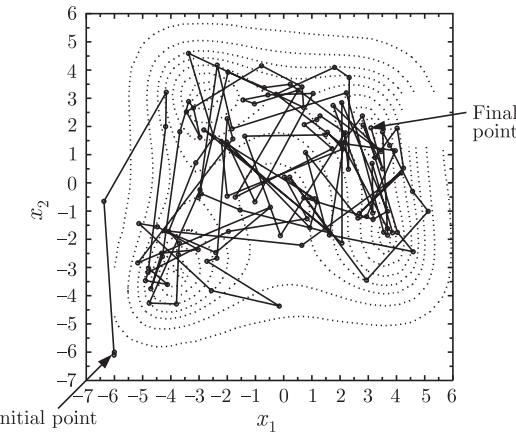
**Table 6.3** Different GA Parameter Settings and Obtained Simulation Results for the Global Minimization Problem

$p_c$	$p_m$	$n$	Maximum generation allowed	Number of runs converged	Average function evaluations
0.8	0.05	20	625	77	426
		30	416	79	484
		40	312	76	650

In all cases the average number of function evaluations required to find a near global minimum solution is less compared to that in a random search method. The table also shows that the performance of GAs (the frequency of convergence) is better than that of the steepest descent method. This leverage of GAs becomes substantial for problems with large number of variables and a large search space. It is worth mentioning here that the parameter settings shown in Table 6.3 are only reasonable values. Carefully choosing the parameters as suggested elsewhere (Goldberg, et al., 1993; Theirens and Goldberg, 1993) may lead to a better performance of genetic algorithms on the above problem.

### 6.3.3 Using Simulated Annealing

Finally, let us consider the simulated annealing procedure. Recall that the simulated annealing procedure begins with an initial point. At every iteration, a neighbouring point is created and compared with the current point. If the new point is better than the current point, the point is accepted; otherwise the new point is accepted using the Metropolis algorithm described in Section 6.2. Figure 6.24 shows the history of intermediate points to global convergence from the starting point  $(-6, -6)^T$ . The temperature is started

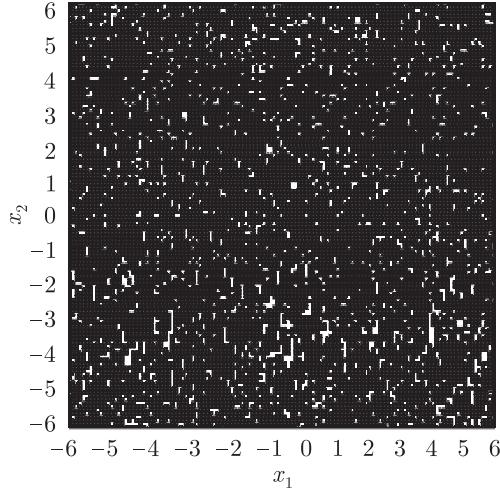


**Figure 6.24** The progress of the simulated annealing algorithm for a simulation run starting from the point  $(-6, -6)^T$ . The algorithm finally converges to the point  $(3, 2)^T$ .

with a value equal to  $T^0 = 100$  and reduced in successive stages  $t$  according to  $T = T^0(1 - t/t_{\max})$ , where the maximum number of stages is fixed at  $t_{\max} = 100$ . In each stage, 50 iterations are performed. The figure shows that the algorithm wanders in the search space before converging to the right solution.

In order to investigate the robustness of the algorithm, we once again divide the search space into 14,641 equally-spaced points (121 points in each

variable) and the simulated annealing algorithm is started from each point independently. If the simulation starting from an initial point converges to a point close to the global minimum, that initial point is marked with a small solid box in Figure 6.25. In each case, the annealing schedule and parameters



**Figure 6.25** Initial points that were successful in converging near the global minimum using the simulated annealing procedure are shown by small solid boxes. About 88 per cent points have converged to the true solution.

as mentioned above are used. The figure shows that simulations starting from most points in the domain have converged to the global optimum. In fact, out of 14,641 points, 12,845 points (about 88 per cent) have converged to the global minimum. The average number of function evaluations required in all successful simulations is 603.

It is worth mentioning here that no effort is made to find the optimal parameter setting (annealing schedule or number of iterations in a stage) for solving the above problem. It may be possible to improve the performance with a better parameter setting. Nevertheless, these results show that GAs and simulated annealing stand as better candidates than the steepest-descent technique in solving multimodal functions to global optimality. Although a simple numerical problem with only four optima is used to compare the performance of nontraditional with traditional optimization methods, the performance of nontraditional methods is expected to be much better than traditional methods in more complex multimodal problems.

#### 6.4 Summary

Two nontraditional search and optimization algorithms motivated by natural principles have been described in this chapter. Genetic algorithms work according to the principles of natural genetics on a population of

string structures representing the problem variables. Three operators—reproduction, crossover, and mutation—are used to create new, and hopefully, better populations. The basic differences of GAs with most of the traditional optimization methods are that GAs use a coding of variables instead of variables directly, a population of points instead of a single point, and stochastic operators instead of deterministic operators. All these features make GA-search robust, allowing them to be applied to a wide variety of problems.

The simulated annealing method mimics the thermodynamic process of cooling of molten metals for achieving the minimum free energy state. The simulated annealing method works with a point and at every iteration creates a point according to the Boltzmann probability distribution. For a slow simulated cooling process, this method guarantees to achieve the global optimum point.

Even though most of this book has dealt with classical optimization methods, those methods are not expected to find the global optimum of a multimodal function at every simulation. Global optimization is important if the function space has a number of optima of which some are local and some global. Since many real-world problems contain multiple optima, traditional methods alone may not be adequate to solve such problems. A function with three local optima and one global optimum has been solved using the steepest descent method and the two nontraditional methods described in this chapter. It has been observed that the steepest descent method could only solve the problem to global optimality about 25 per cent of the time. Genetic algorithms and simulated annealing, on the other hand, could solve the problem to global optimality most of the time.

## REFERENCES

- Aarts, E. and Korst, J. (1989). *Simulated Annealing and Boltzmann machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Chichester: Wiley.
- Babu, B. and Jehan, M. L. (2003). Differential evolution for multi-objective optimization. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003)*, volume 4, pp. 2696–2703, Canberra, Australia: IEEE Press.
- Belew, R. K. and Booker, L. B., Ed. (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms*. San Mateo, California: Morgan Kaufmann.
- Brest, J. (2009). Constrained real-parameter optimization with  $\varepsilon$ -self-adaptive differential evolution. In Mezura-Montes, E., Ed., *Constraint-Handling in Evolutionary Optimization*, volume 198, pp. 73–93. Heidelberg: Springer.

- Callahan, K. J. and Weeks, G. E. (1992). Optimal design of composite laminates using genetic algorithms. *Composite Engineering*, **2**(3), 149–160.
- Coello, C. A. C. and Lechuga, M. S. (2002). MOPSO: A proposal for multiple objective particle swarm optimization. In *Congress on Evolutionary Computation (CEC'2002)*, volume 2, pp. 1051–1056, Piscataway, New Jersey: IEEE Service Center.
- Coello, C. A. C. and Toscano, G. (2000). A micro-genetic algorithm for multi-objective optimization. Technical Report Lania-RI-2000-06, Laboratoria Nacional de Informatica Avanzada, Xalapa, Veracruz, Mexico.
- Corne, D. W., Knowles, J. D., and Oates, M. (2000). The Pareto envelope-based selection algorithm for multiobjective optimization. In *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature VI (PPSN-VI)*, pp. 839–848.
- Davis, L., Ed. (1991). *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.
- Davis, T. and Principe, J. C. (1991). A simulated annealing like convergence theory for the simple genetic algorithm. In R. K. Belew and L. B. Booker, Eds. (1991), *op. cit.*, pp. 174–181.
- Deb, K. (2001). *Multi-objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK.
- Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, **186**(2–4), 311–338.
- Deb, K. (1991). Optimal design of a welded beam structure via genetic algorithms. *AIAA Journal*, **29**(11), 2013–2015.
- Deb, K. (1989). Genetic algorithms in multimodal function optimization, *Master's Thesis*, (TCGA Report No. 89002). Tuscaloosa: University of Alabama.
- Deb, K. and Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems*, **9**(2), 115–148.
- Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2002). A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**(2), 182–197.
- Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI (PPSN-VI)*, pp. 849–858.
- Deb, K. and Datta, R. (2010). A fast and accurate solution of constrained optimization problems using a hybrid bi-objective and penalty function approach. In *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI-2010)*, pp. 165–172.

- Eshelman, L. J. and Schaffer, J. D. (1993). Real-coded genetic algorithms and interval schemata. In D. Whitley, Ed. (1993), *op. cit.*, pp. 187–202.
- Fonseca, C. M. and Fleming P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In S. Forrest, Ed., (1993), *op. cit.*, pp. 416–423.
- Forrest, S., Ed. (1993). *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Mateo, California: Morgan Kaufmann.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley.
- Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning. *Dissertation Abstracts International*, **44**(10), 3174B. (University Microfilms No. 8402282).
- Goldberg, D. E., Deb, K., and Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. In R. Manner and B. Manderick, Eds., *Parallel Problem Solving from Nature II*, North-Holland, Amsterdam, pp. 37–46.
- Goldberg, D. E., Deb, K., and Theirens, D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of SICE*, **32**(1), 10–16.
- Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results, *Complex Systems*, **3**, 493–530.
- Goldberg, D. E., Milman, K., and Tidd, C. (1992). Genetic algorithms: A bibliography (IlliGAL Report No. 92008). Urbana: University of Illinois at Urbana-Champaign. Illinois Genetic Algorithms Laboratory.
- Goldberg, D. E. and Richardson, J. (1987): Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, Ed. (1987), *op. cit.*, pp. 41–49.
- Gravel, M., Price, W. L., and Gagné, C. (2002). Scheduling continuous casting of aluminum using a multiple objective ant colony optimization metaheuristic. *European Journal of Operational Research*, **143**(1), 218–229.
- Grefenstette, J. J., Ed. (1985). *Proceedings of an International Conference on Genetic Algorithms and Their Applications*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Grefenstette, J. J., Ed. (1987). *Proceedings of the Second International Conference on Genetic Algorithms*. Hillsdale, New Jersey: Lawrence Erlbaum Associates.
- Hajela, P. (1990). Genetic search: An approach to the nonconvex optimization problem. *AAAI Journal*, **28**, 1205–1210.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.

- Horn, J. and Nafpliotis N. (1993). Multiobjective optimization using niched Pareto genetic algorithms (IlliGAL Report No. 93005). Urbana: University of Illinois at Urbana-Champaign. Illinois Genetic Algorithms Laboratory.
- Jenkins, W. M. (1991). Towards structural optimization via the genetic algorithm. *Computers and Structures*, **40**(5), 1321–1327.
- Kargupta, H., Deb, K., and Goldberg, D. E. (1992). Ordering genetic algorithms and deception. In R. Manner and B. Manderick, Eds., *Parallel Problem Solving from Nature II*, pp. 47–56.
- Kirkpatrick, S., Gelatt., C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**(4598), 671–680.
- Knuth, D. E. (1981). *The Art of Computer Programming*, 2nd ed., Vol. 2. Reading, Mass.: Addison-Wesley.
- Loughlin, D. H. and Ranjithan, S. (1997). The neighbourhood constraint method: A multiobjective optimization technique. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 666–673.
- Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**, 1087–1092.
- McMullen, P. R. (2001). An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives. *Artificial Intelligence in Engineering*, **15**, 309–317.
- Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin: Springer-Verlag.
- Mostaghim, S. and Teich, J. (2003). Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO). In *2003 IEEE Swarm Intelligence Symposium Proceedings*, pp. 26–33, Indianapolis, Indiana, USA: IEEE Service Center.
- Powell, D. J., Tong, S. S., and Skolnick, M. M. (1989). EnGENEous domain independent, machine learning for design optimisation. In J. D. Schaffer, Ed. (1989), *op. cit.*, pp. 151–159.
- Radcliffe, N. J. (1990). *Genetic neural networks on MIMD computers. Doctoral Dissertation*. Edinburgh: University of Edinburgh.
- Rajeev, S. and Krishnamoorthy, C. S. (1992). Discrete optimization of structures using genetic algorithms. *Journal of Structural Engineering, ASCE*, **118**(5), 1233–1250.
- Rawlins, G. J. E., Ed. (1991). *Foundations of Genetic Algorithms*. San Mateo, California: Morgan Kaufmann.
- Sasaki, D., Morikawa, M., Obayashi, S., and Nakahashi, K. (2001). Aerodynamic shape optimization of supersonic wings by adaptive range multiobjective genetic algorithms. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*, pp. 639–652.

- Schaffer, J. D., Ed. (1989). *Proceedings of the Third International Conference on Genetic Algorithms*. San Mateo, California: Morgan Kaufmann.
- Schaffer, J. D. (1984). *Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithms*. Doctoral dissertation, Vanderbilt University, Electrical Engineering, Tennessee. (TCGA file No. 00314).
- Srinivas, N. (1994). *Multiobjective Optimization using Nondominated Sorting in Genetic Algorithms*. Masters thesis, Indian Institute of Technology, Kanpur.
- Srinivas, N. and Deb, K. (1995). Multiobjective function optimization using nondominated sorting genetic algorithms. *Evolutionary Computation*, **2**(3), 221–248.
- Takahama, T. and Sakai, S. (2006). Constrained optimization by the  $\epsilon$ -constrained differential evolution with gradient-based mutation and feasible elites. In *2006 IEEE Congress on Evolutionary Computation*, pp. 1–8. Piscataway: IEEE Press.
- Thierens, D. and Goldberg, D. E. (1993). Mixing in genetic algorithms. In S. Forrest, Ed. (1993), *op. cit.*, pp. 38–45.
- Veldhuizen, D. V. and Lamont, G. B. (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary Computation Journal* **8**(2), 125–148.
- Vose, M. D. and Liepins, G. E. (1991). Punctuated equilibria in genetic search. *Complex Systems*, **5**(1), 31–44.
- Whitley, D., Ed. (1993). *Foundations of Genetic Algorithms II*. San Mateo, California: Morgan Kaufmann.
- Wright, A. (1991). Genetic algorithms for real parameter optimization. In G. J. E. Rawlins, Ed. (1991), *op. cit.*, pp. 205–220.
- Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In Giannakoglou, K. C., Tsahalis, D. T., Périaux, J., Papailiou, K. D., and Fogarty, T., Eds., *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pp. 95–100, Athens, Greece. International Center for Numerical Methods in Engineering (CIMNE).

## PROBLEMS

**6-1** In a three-variable problem, the following variable bounds are specified.

$$-5 \leq x \leq 10,$$

$$0.001 \leq y \leq 0.005,$$

$$1(10^3) \leq z \leq 1(10^4).$$

What should be the minimum string length of any point  $(x, y, z)^T$  coded in binary string to achieve the following accuracy in the solution:

- (i) Two significant digits.
- (ii) Three significant digits.

**6-2** Repeat Problem 6-1 if ternary strings (with three alleles 0, 1, and 2) are used instead of binary strings.

**6-3** We would like to use a binary-coded genetic algorithm to solve the following NLP problem:

$$\text{Minimize } (x_1 - 1.5)^2 + (x_2 - 4)^2$$

subject to

$$4.5x_1 + x_2^2 - 22.5 \leq 0,$$

$$2x_1 - x_2 - 1 \geq 0,$$

$$0 \leq x_1, x_2 \leq 4.$$

We decide to have two and three decimal places of accuracy for variables  $x_1$  and  $x_2$ , respectively.

- (i) Find the optimum solution graphically and clearly indicate the optimum solution on the plot.
- (ii) At least how many bits are required for coding the variables?
- (iii) Use the penalty-parameter-less fitness assignment procedure to compute the fitness of the following population members: (i) (2, 1) (ii) (6, 2) (iii) (0, 3) (iv) (2, 8) (v) (3, 0) (vi) (1, -1) (vii) (2.5, 2).

**6-4** In a 10-bar truss-structure design optimization problem, the cross-section of each member is a variable. It is decided that the members are to be made either square or circular. If a square cross-section is to be used, the side length of the square is a design variable. In the case of a circular member, the diameter of the cross-section is a design variable. Either the side length of the square or the diameter of the circle can take a value in the range (5, 30) mm. What is the minimum overall string length for a binary coding of the problem to achieve a solution up to three decimal places of accuracy?

**6-5** Problem 6-4 needs to be modified not only to find the optimal cross-sections but also to find the optimal topology of the truss with a maximum of 10 members. Design a coding for the problem using binary strings. One way to code the problem is to assign a value 1 if the member is present, a 0 otherwise. Choose a random string and draw a rough sketch of the truss indicating the size and shape of each member.

**6-6** We would like to use genetic algorithms to solve the following NLP problem:

$$\text{Minimize } (x_1 - 1.5)^2 + (x_2 - 4)^2$$

subject to

$$4.5x_1 + x_2^2 - 18 \leq 0,$$

$$2x_1 - x_2 - 1 \geq 0,$$

$$0 \leq x_1, x_2 \leq 4.$$

We decide to have three and two decimal places of accuracy for variables  $x_1$  and  $x_2$ , respectively.

- (i) How many bits are required for coding the variables?
- (ii) Write down the fitness function which you would be using in reproduction.
- (iii) Which schemata represent the following regions according to your coding:
  - (a)  $0 \leq x_1 \leq 2$ ,
  - (b)  $1 \leq x_1 \leq 2$ , and  $x_2 > 2$ ?

**6-7** In a genetic algorithm simulation, the following population is observed at generation four:

Strings	Fitness
0101101	10
1001001	8
1010110	14
0111110	20
1110111	2
0111100	12

If roulette-wheel selection, single-point crossover with probability 0.9, and a bit-wise mutation with a probability 0.01 are used, how many copies of the schema (\*1\*11\*\*) are expected in generation five?

**6-8** Consider the following population of binary strings for a maximization problem:

Strings	Fitness
01101	5
11000	2
10110	1
00111	10
10101	3
00010	100

Find out the expected number of copies of the best string in the above population in the mating pool under

- (i) roulette-wheel selection.
- (ii) binary tournament selection without replacement.

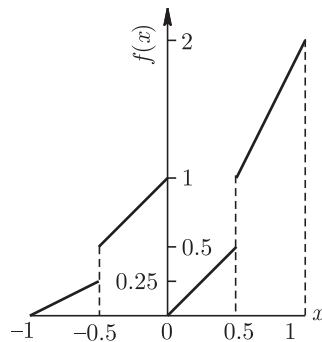
If only the reproduction operator is used, how many generations are required before the best individual occupies the complete population under each selection operator?

**6-9** Consider the following population of five strings, having three choices in each place:

Strings	Fitness (to be maximized)
♡♣♦♣♣♡	6
◊♦◊♡♣♣	1
♡◊◊♣♦◊♡	8
♣♡♣♦♡◊	2
◊♣♡♣♦◊♡	6

- (i) How many copies would ( $\heartsuit * * * * *$ ) have after one iteration of a proportionate selection operation, single-point crossover operator with  $p_c = 0.8$ , and bit-wise mutation operator with  $p_m = 0.1$ ?
- (ii) What proportion of the entire search space is represented by the schema ( $\clubsuit * \diamond * * *$ )?
- (iii) Realizing that the above representation uses a ternary coding ( $\clubsuit \rightarrow 0$ ,  $\diamond \rightarrow 1$ , and  $\heartsuit \rightarrow 2$ ), how many discrete regions in the decoded integer space are represented by the schema ( $\clubsuit * * \diamond \heartsuit *$ )?

**6-10** A piece-wise linear single-variable ( $x \in [-1, 1]$ ) problem shown in Figure 6.26 needs to be maximized. At a discontinuous point, the larger function value is assigned to the point.



**Figure 6.26** For Problem 6-10.

- (i) Locate all the local maximum points.
- (ii) To represent a solution, 10 bits are used. Clearly indicate the region represented by the schema  $*11*****$ .

- (iii) Starting with a uniformly distributed initial population, calculate the proportion of points representing the above schema in the population after two iterations of binary tournament selection operator alone.

**6-11** From a schema processing point of view, derive the schema theorem for a ternary string coding (three options at each position) under roulette-wheel selection, single-point crossover and allele-wise mutation (one allele value changing to any of the two other allele values with equal probability).

**6-12** Consider the following single-variable maximization problem with  $x \in [0, 1]$  and having global maximum at  $x = 1$  and a local maximum at  $x = 0.25$  that is to be solved using a binary-coded GA with a very large-size string coding for  $x$ :

$$f(x) = \begin{cases} 3.2x, & \text{if } 0 \leq x \leq 0.25, \\ 3.2(0.5 - x), & \text{if } 0.25 \leq x \leq 0.5, \\ 0, & \text{if } 0.5 \leq x \leq 0.875, \\ 8(x - 0.875), & \text{if } 0.875 \leq x \leq 1. \end{cases}$$

- (i) If a minimum of 5 points are needed in each partition to have a representative average schema fitness, what is the minimum population size needed to solve the problem to global optimality from a schema processing point of view?
- (ii) If a sharing method is used to find both local and global maxima in a single GA run with the proportionate selection operator and a minimum of 36 points are needed in each optimum to take care of stochasticities, what is the minimum population size needed to maintain both maxima in the population?

**6-13** In the mutation operator in a real-parameter GA, a neighbouring solution is created with the following probability distribution ( $\sigma$  is a user-supplied parameter):

$$p(x) = \frac{a}{\sigma^2 + x^2}, \quad x \in \Re.$$

- (i) What must be the value of  $a$ , in order to have a valid probability distribution?
- (ii) The current point is  $x^{(t)} = 10.0$ . In order to perform mutation with  $\sigma = 2.0$ , a random number 0.723 is used. What is the mutated point  $x'$ ?

**6-14** Using real-coded GAs with simulated binary crossover (SBX) having  $\eta = 2$ , find the probability of creating children solutions in the range  $0 \leq x \leq 1$

with two parents  $x^{(1)} = 0.5$  and  $x^{(2)} = 3.0$ . Recall that for SBX operator, the children solutions are created using the following probability distribution:

$$\mathcal{P}(\beta) = \begin{cases} 0.5(\eta + 1)\beta^\eta, & \text{if } \beta \leq 1; \\ 0.5(\eta + 1)/\beta^{\eta+2}, & \text{if } \beta > 1. \end{cases}$$

where  $\beta = |(c_2 - c_1)/(p_2 - p_1)|$  and  $c_1$  and  $c_2$  are children solutions created from parent solutions  $p_1$  and  $p_2$ .

**6-15** Using real-coded GAs with simulated binary crossover (SBX) having  $\eta = 2$ , on an average, what proportion of offsprings will lie in the range  $2.5 \leq x \leq 5.0$  with two parents  $x^{(1)} = 2.0$  and  $x^{(2)} = 4.0$ ?

**6-16** Two parents are shown below (unbounded  $x_i$ ):

$$\mathbf{x}^{(1)} = (12.0, 5.0)^T, \quad \mathbf{x}^{(2)} = (5.0, 8.0)^T.$$

Calculate the probability of finding a child in the range  $x_i \in [0.0, 5.0]$  for  $i = 1, 2$  using the simulated binary crossover (SBX) operator with  $\eta_c = 2$ .

**6-17** Apply the polynomial mutation operator (with  $\eta_m = 5.0$ ) to create a mutated child of the solution  $x^{(t)} = 6.0$  using a random number 0.675. Assume  $x \in [0, 10]$ .

**6-18** Find the probability of creating an offspring in the range  $x \in [0.8, 1.0]$  from the parent  $p = 0.9$  using polynomial mutation operator with  $\eta_m = 20$  for a bounded problem with  $x \in [0, 1]$ .

**6-19** Calculate the overall probability of creating a child in the range  $0.0 \leq (x_1, x_2) \leq 0.2$  by mutating a parent  $(0.5, 0.3)^T$  for a two-variable unbounded problem using polynomial mutation with  $\eta_m = 10$ . Assume a variable-wise mutation probability of  $p_m = 0.5$ .

**6-20** A cantilever beam of circular cross-section (diameter  $d$ ) has to be designed for minimizing the cost of the beam. The beam is of length  $l = 0.30$  m and carries a maximum of  $F = 1$  kN force at the free end. The beam material has  $E = 100$  GPa,  $S_y = 100$  MPa, and density  $\rho = 7866$  kg/m<sup>3</sup>. The material costs Rs. 20 per kg. There are two constraints: (a) Maximum stress in the beam ( $\sigma = 32Fl/(\pi d^3)$ ) must not be more than the allowable strength  $S_y$ , (b) Maximum deflection of the beam must not be more than 1 mm. The deflection at the free end due to the load  $F$  is  $\delta = 64Fl^3/(3\pi Ed^4)$ . The volume of the beam is  $\pi d^2 l/4$ . The initial population contains the following solutions:  $d = 0.06, 0.1, 0.035, 0.04, 0.12$ , and  $0.02$  m.

- (i) For each of six solutions, calculate cost in rupees and determine its feasibility in a tabular format.
- (ii) Determine the fitness of each solution using penalty-parameter-less constraint handling strategy. Which is the best solution in the above population?

**6-21** For the following two-objective optimization problem,

$$\begin{aligned} & \text{Minimize } x^3 - x, \\ & \text{Minimize } x^2, \\ & \text{subject to } x \geq 0, \end{aligned}$$

a GA considers the following seven population members:  $x = -1, 0, 1, -2, 0.5, 2, 0.2$ . Rank these solutions according to an increasing order of constraint domination.

**6-22** We intend to solve the following two-objective optimization problem with  $S$  being the feasible solution set:

$$\begin{aligned} & \text{Minimize } f(x), \\ & \text{Maximize } f(x), \end{aligned}$$

subject to

$$x \in \mathcal{S}.$$

Individual optimal solutions of the above two objectives are  $x_{\min}^*$  and  $x_{\max}^*$ , respectively. Which solution(s) in the search space  $S$  constitute the Pareto-optimal front?

**6-23** For the following problem

$$\begin{aligned} & \text{Minimize } f_1(\mathbf{x}) = x_1, \\ & \text{Minimize } f_2(\mathbf{x}) = x_2, \end{aligned}$$

subject to

$$\begin{aligned} g_1(x) &\equiv x_1^2 + x_2^2 \leq 1, \\ g_2(x) &\equiv (x_1 - 1)^2 + (x_2 - 1)^2 \leq 1, \end{aligned}$$

identify the Pareto-optimal set when

- (i) both  $f_1$  and  $f_2$  are minimized,
- (ii)  $f_1$  is minimized and  $f_2$  is maximized,
- (iii)  $f_1$  is maximized and  $f_2$  is minimized,
- (iv) both  $f_1$  and  $f_2$  are maximized.

Draw a neat sketch of the feasible region.

**6-24** Identify the non-dominated set from the following points (all objectives are to be minimized):

Soln. Id.	$f_1$	$f_2$	$f_3$
1	2	3	1
2	5	1	10
3	3	4	10
4	2	2	2
5	3	3	2
6	4	4	5

**6-25** Consider the following parent and offspring populations for a problem of minimizing the first objective and maximizing the second objective:

Soln.	$P_t$		$Q_t$	
	$f_1$	$f_2$	$f_1$	$f_2$
1	5.0	2.5	a	3.5 0.0
2	1.0	1.0	b	2.2 0.5
3	1.5	0.0	c	5.0 2.0
4	4.5	1.0	d	3.0 3.0
5	3.5	2.0	e	3.0 1.5

Create  $P_{t+1}$  using NSGA-II.

**6-26** Consider the problem:

$$\text{Minimize } f_1(\mathbf{x}) = (x_1 - 2)^2 + x_2^2,$$

$$\text{Maximize } f_2(\mathbf{x}) = 9x_1 - x_2^2,$$

subject to

$$1.0 - (x_1^2 + x_2^2)/225 \geq 0,$$

$$(-x_1 + 3x_2)/10 - 1.0 \geq 0,$$

and the following points in a population:

$$\mathbf{x}^{(1)} = (0, 0)^T, \quad \mathbf{x}^{(2)} = (5, 10)^T, \quad \mathbf{x}^{(3)} = (10, -15)^T,$$

$$\mathbf{x}^{(4)} = (-11, 0)^T, \quad \mathbf{x}^{(5)} = (10, 10)^T, \quad \mathbf{x}^{(6)} = (0, 15)^T.$$

- (i) If constraints are not used, sort the population in increasing level of non-domination.
- (ii) By considering the constraints, sort the population in increasing level of constraint non-domination.

**6-27** In order to solve the maximization problem

$$f(x) = \begin{cases} 2x, & \text{if } x \leq 0.3 \\ 1.5 - 3x, & \text{if } 0.3 < x \leq 0.5 \\ 5x - 2.5, & \text{if } 0.5 < x \leq 0.8 \\ 2.4 - 2x, & \text{if } x > 0.8 \end{cases}$$

using simulated annealing, we begin with an initial point  $x^{(0)} = 0.2$ . A standard normal probability distribution is used to create a neighbouring point.

- (i) Calculate the probability of creating a point in the global basin.
  - (ii) If a temperature  $T = 10$  is used, what is the overall probability of accepting a point in the global basin?

## **COMPUTER PROGRAM**

A computer program implementing genetic algorithms for optimizing multivariable problems is presented below. The objective function is coded in the function **funct**. The program is written in FORTRAN and part of the program is adapted from David E. Goldberg's pascal code (Goldberg, 1989). The program uses dynamic memory allocation for storing chromosomes, variables, fitness values, and other program parameters. Thus, if the program finds that the allocated memory is not enough for the entire simulation of GAs, the program terminates without performing any further computation. The allocated memory (specified by **parameter** declaration in the beginning of the code) can then be increased and the program can be compiled and run again. This code is developed to handle maximization problems. In order to optimize minimization problems, a transformation suggested in Equation (6.2) can be used.

```
program sga
*****
c      *
c      *          GENETIC ALGORITHMS      *
c      *          *
c      *****
c Developed by Kalyanmoy Deb
c           Indian Institute of Technology, Kanpur
c (Adapted in Fortran from David E. Goldberg's pascal
c   code)
c All rights reserved. This listing is for personal use.
*****
```

```

c This routine finds the optimum point of a constrained
c optimization problem using genetic algorithms.
c Code your function in subroutine funct
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      parameter (max1=5000,max2=10000,max3=5000)
      integer*1 a(max2)
      dimension b(max1)
      integer   c(max3)
c..... common blocks .....
c....for GA parameters
      common/sgaparam/ipopsizelchrom,maxgen,ncross,
      -          nmute,pcross,pmute,npnparam
c....for statistical results
      common/statist/igen,avg,amax,amin,sumfitness
c....for random variables routines
      common/randvar/oldrand,jrand
c....for best individual
      common/bestpt/ibest,bestfit,igenbst
c....for other constant
      common/consts/fix
c..... fixed array for random number generator .....
      real oldrand(55), fix
      integer jrand
c..... read all GA input parameters first .....
      call initdata
c..... dynamic memory allocation .....
c.....array for all old chromosomes
      na1 = 1
c.....array for all new chromosomes
      na2 = na1 + ipopsizelchrom
c.....array for a dummy chromosome
      na3 = na2 + ipopsizelchrom
c.....total memory for chromosomes
      na4 = na3 + lchrom
      if (na4 .gt. max1) then
        write(*,9010) max1,na4
9010    format(' Allocated memory of ',i8,
      -       ' is not enough',/, ' Increase to at least',i8)
        stop
      endif
c.....array for all x parameters of oldpop
      nb1 = 1
c.....array for all x parameters of newpop
      nb2 = nb1 + ipopsizelnpnparam
c.....array for all fitness values of oldpop
      nb3 = nb2 + ipopsizelnpnparam

```

```

c.....array for all fitness values of newpop
    nb4 = nb3 + ipopsiz
c.....array for fraction (used in selection)
    nb5 = nb4 + ipopsiz
c.....array for factor (total points in each x)
    nb6 = nb5 + ipopsiz
c.....array for a dummy x vector
    nb7 = nb6 + ipopsiz
c.....array for lower bound of variables
    nb8 = nb7 + ipopsiz
c.....array for upper bound of variables
    nb9 = nb8 + nparam
c.....array for best solution
    nb10 = nb9 + nparam
c.....total memory for real variables
    nb11 = nb10 + nparam
    if (nb11 .gt. max2) then
        write(*,9010) max2,nb11
        stop
    endif
c.....array for lsubstr (each string length)
    nc1 = 1
c.....array for choices (used in selection)
    nc2 = nc1 + nparam
c.....total memory for integer variables
    nc3 = nc2 + ipopsiz
    if (nc3 .gt. max3) then
        write(*,9010) max3,nc3
        stop
    endif
c..... main program begins here .....
    igen = 0
c.....create initial population and evaluate ====
    call initialize(a(na1),b(nb1),b(nb3),b(nb6),
    -      c(nc1),b(nb8),b(nb9),b(nb7),a(na3),b(nb10))
c.....main generation loop begins =====
    10 igen = igen + 1
c.....perform one generation of GA operators ====
    c      contains reproduction, crossover and mutation
    call generation(a(na1),a(na2),b(nb2),b(nb3),
    -          b(nb4),c(nc2),b(nb5),b(nb7),b(nb6),
    -          c(nc1),b(nb8),b(nb9),a(na3))
c.....calculate statistics of the new population ===
    call stats(b(nb4))
c.....print the simulation results =====
    call report(b(nb1),b(nb2),b(nb3),b(nb4),b(nb10))

```

```

c.....finally copy new population to old population ==
    call copynewtoold(a(na1),a(na2),b(nb1),b(nb2),
    -                  b(nb3),b(nb4))
c.....put your termination criteria =====
    if (igen .lt. maxgen) go to 10
    stop
    end
c..... end of main program ......

c      ***** objective function for optimization *****
function funct(x)
common/sgaparam/ipopsizelchrom,maxgen,ncross,
-                      nmute,pcross,pmute,nparam
real x(nparam)
c.....code your function here
obj = (x(1)**2+x(2)-11)**2+(x(1)+x(2)**2-7)**2
c.....this transformation is for minimization problems
obj = 1.0/(1.0+obj)
funct = obj
return
end

c      ***** initializes GA parameters *****
subroutine initdata
common/sgaparam/ipopsizelchrom,maxgen,ncross,
-                      nmute,pcross,pmute,nparam
write(*,*)'
write(*,*)'          Genetic Algorithms in FORTRAN'
write(*,*)'          Kalyanmoy Deb'
write(*,*)'          All rights reserved'
write(*,*)'

c.....read all GA parameters
write(*,*)'** GA data entry and initialization **'
write(*,*)'enter number of parameters -----> '
read(*,*) nparam
write(*,*)'enter total string length -----> '
read(*,*) lchrom
write(*,*)'enter population size -----> '
read(*,*) ipopsizelchrom
write(*,*)'enter max. generation -----> '
read(*,*) maxgen
write(*,*)'enter crossover probability -----> '
read(*,*) pcross
write(*,*)'enter mutation probability -----> '
read(*,*) pmutenparam
nmute = 0

```

```

ncross = 0
return
end

c      ***** initializes the data, population and
c          statistics variables                      *****
c subroutine initialize(olddchrom,oldx,oldfit,factor,
-           lsubstr,alow,ahigh,dummyx,dummychr,bestsol)
common/sgaparam/ipopsizelchrom,maxgen,ncross,
-           nmute,pcross,pmute,npam
common/consts/fix
common/bestpt/ibest,bestfit,igenbst
integer*1 oldchrom(ipopsizelchrom)
integer*1 dummychr(lchrom)
real oldx(ipopsizelparam),oldfit(ipopsizelparam)
real factor(npam), dummyx(npam)
real alow(npam), ahigh(npam)
real bestsol(npam)
integer lsubstr(npam)
lchrom = 0
c.....read the string coding information
do 10 i = 1,npam
    write(*,9040) i
9040  format(' enter string length for variable',i3)
        read(*,*) lsubstr(i)
        write(*,9041) i
9041  format(' enter minimum and maximum bound on',
            ' variable',i3)
        read(*,*) alow(i),ahigh(i)
        factor(i) = 2.0**float(lsubstr(i))-1.0
        lchrom = lchrom + lsubstr(i)
10   continue
c.....initialize the random number generator
    call randomize
c.....create the initial population
    call initpop(olddchrom,oldx,oldfit,dummyx,dummychr,
-           lsubstr,alow,aheight,factor)
c.....calculate initial population statistics
    call stats(oldfit)
c.....assign the best individual
    bestfit = oldfit(ibest)
    do 12 j = 1,npam
12      bestsol(j) = oldx(ibest,j)
    igenbst = 0
c.....report the initial population statistics
    call initreport

```

```

c.....initialize format specification
      return
      end

c      ***** prints report of initial population *****
      subroutine initreport
      common/sgaparam/ipopsiz, lchrom, maxgen, ncross,
      -          nmut, pcross, pmut, nparam
      common/statist/igen, avg, amax, amin, sumfitness
      common/consts/fix
      write(*,*)'
      write(*,*)'          Genetic Algorithms in FORTRAN'
      write(*,*)'          Kalyanmoy Deb'
      write(*,*)'          All rights reserved'
      write(*,*)'
      write(*,*)'          GA parameters '
      write(*,*)'-----',
      write(*,*)'

c.....print the GA parameters
      write(*,9050) ipopsiz
      write(*,9051) lchrom
      write(*,9052) maxgen
      write(*,9053) pcross
      write(*,9054) pmut
      write(*,9055) fix
      write(*,*)'

c.....print the initial statistics
      write(*,9056) amax
      write(*,9057) avg
      write(*,9058) amin
      write(*,*)'

9050 format(' population size      = ',i4)
9051 format(' chromosome length     = ',i4)
9052 format(' max. # of generations = ',i4)
9053 format(' crossover probability = ',f8.3)
9054 format(' mutation probabiltiy = ',f8.3)
9055 format(' seed random number   = ',f8.3)
9056 format(' maximum fitness        = ',f10.5)
9057 format(' average fitness         = ',f10.5)
9058 format(' minimum fitness         = ',f10.5)

      return
      end

c.....***** generates initial random population *****
      subroutine initpop(chrom,oldx,fitness,x,chr,
      -          lsubstr,alow,ahigh,factor)

```

```

      common/sgaparam/ipopsizelchrom,maxgen,ncross,
      -          nmutepcross,pmute,npnparam
      integer*1 chrom(ipopsizelchrom), chr(lchrom)
      real oldx(ipopsizelnpnparam), fitness(ipopsizelchrom)
      real alow(npnparam), ahigh(npnparam), factor(npnparam)
      integer lsubstr(npnparam)
      real x(npnparam)
      do 121 j = 1,ipopsizelchrom
         do 122 j1 = 1,lchrom
c..... create random strings of 1 and 0
            chr(j1) = iflip(0.5)
            chrom(j,j1) = chr(j1)
122      continue
c..... calculate x values from the string
            call decodevars(chr,x,lsubstr,alow,ahigh,
            -                  factor)
            do 123 j1 = 1,npnparam
               oldx(j,j1) = x(j1)
123      continue
c..... calculate the fitness of the string
            fitness(j) = funct(x)
121      continue
      return
      end

c ***** update statistics (avg, min, max, sum) *****
      subroutine stats(fitness)
      common/sgaparam/ipopsizelchrom,maxgen,ncross,
      -          nmutepcross,pmute,npnparam
      common/statist/igen,avg,amax,amin,sumfitness
      common/bestpt/ibest,bestfit,igenbst
      real fitness(ipopsizelchrom)
c..... assign first individual as min, max, and best
      sumfitness = fitness(1)
      amin = fitness(1)
      amax = fitness(1)
      ibest= 1
      do 10 j = 2,ipopsizelchrom
         sumfitness = sumfitness + fitness(j)
c..... then check if any other is better
         if(fitness(j) .gt. amax) then
            amax = fitness(j)
            ibest= j
         endif
c..... keep a track of the minimum fitness
         if(fitness(j) .lt. amin) amin = fitness(j)

```

```

10    continue
c.....compute the average fitness of the population
      avg = sumfitness/float(ipopsiz)
      return
      end

c      ***** perform one generation of reproduction,
c          crossover and mutation operations      *****
c subroutine generation(oldchr,newchr,newx,oldfit,
-          newfit,choices,fraction,x,factor,lsubstr,
-          alow,ahigh,chr)
c common/sgaparam/ipopsizelchrom,maxgen,ncross,
-          nmute,pcross,pmute,nparam
c integer*1 oldchr(ipopsizelchrom)
c integer*1 newchr(ipopsizelchrom), chr(lchrom)
c real newx(ipopsizenparam), x(nparam)
c real oldfit(ipopsiz), newfit(ipopsiz)
c integer choices(ipopsiz), lsubstr(nparam)
c real fraction(ipopsiz)
c real alow(nparam), ahigh(nparam), factor(nparam)
c.....primer for the stochastic remainder
c.....roulette wheel selection
      call preselect(oldfit,choices,fraction,nremain)
c.....initialize the individual counter
      j = 1
c.....select mates for reproduction
      181 mate1 = iselect2(choices,nremain)
      mate2 = iselect2(choices,nremain)
c.....perform crossover using the selected mates
      call crossover(j,mate1,mate2,oldchr,newchr)
c.....compute real values of the first child string
      do 20 j1 = 1,lchrom
          chr(j1) = newchr(j,j1)
20    continue
      call decodevars(chr,x,lsubstr,alow,ahigh,factor)
      do 10 j1 = 1,nparam
          newx(j,j1) = x(j1)
10    continue
c.....compute fitness of the first child string
      newfit(j) = funct(x)
c.....compute real values of the second child string
      do 22 j1 = 1,lchrom
          chr(j1) = newchr(j+1,j1)
22    continue
      call decodevars(chr,x,lsubstr,alow,ahigh,factor)

```

```

      do 11 j1 = 1,npParam
         newx(j+1,j1) = x(j1)
11   continue
c.....compute fitness of the second child string
      newfit(j+1) = funct(x)
c.....update the individual count in the population
      j = j + 2
c.....if the population is already filled up, quit
      if(j .le. ipopsizE) go to 181
      return
      end
c   *** primer for the stochastic rem. RW selection
c       updates array 'choices' (mate pool) *****
      subroutine preselect(oldfit,choices,fraction,
-           nremain)
      common/sgaparam/ipopsizE,lchrom,maxgen,ncross,
-           nmute,pcross,pmute,npParam
      common/statist/igen,avg,amax,amin,sumfitness
      real oldfit(ipopsizE)
      integer choices(ipopsizE)
      real fraction(ipopsizE)
      integer*1 winner
      j=0
      k=0
141   j=j+1
      expected = oldfit(j) / avg
c.....assign mantissa number of copies first
      jassign = ifix(expected)
c.....do roulette wheel operation with decimals
      fraction(j) = expected - jassign
142   if(jassign .lt. 0)  go to 143
      k = k + 1
      jassign = jassign - 1
      choices(k) = j
      go to 142
143   if(j .lt. ipopsizE) go to 141
      j = 0
144   if(k .ge. ipopsizE) go to 145
      j = j + 1
      if(j .gt. ipopsizE) j = 1
      if(fraction(j) .gt. 0.0) then
c..... check if the wheel points to the individual
         winner = iflip(fraction(j))
         if(winner .eq. 1) then
            k = k + 1
            choices(k) = j

```

```

        fraction(j) = fraction(j) - 1.0
    endif
endif
go to 144
c.....set nremain as the population size
145 nremain = ipopsiz
      return
end

c *** Stochastic remaindar roulette wheel selection
c     selects individuals from mating pool ***
function iselect2(choices,nremain)
common/sgaparam/ipopsiz,1chrom,maxgen,ncross,
-           nmute,pcross,pmute,nparam
integer choices(ipopsiz)
jpick = irnd(1,nremain)
iselect2 = choices(jpick)
choices(jpick) = choices(nremain)
nremain = nremain - 1
return
end

c ***** simple roulette wheel selection *****
c             (not used here)
function select(fitness)
common/sgaparam/ipopsiz,1chrom,maxgen,ncross,
-           nmute,pcross,pmute,nparam
common/statist/igen,avg,amax,amin,sumfitness
real fitness(ipopsiz)
partsum = 0.0
j = 0
rand = random() * sumfitness
151 j = j + 1
partsum = partsum + fitness(j)
c.....check if the wheel points to the individual j
if((partsum.ge.rand).or.(j.eq.ipopsiz)) go to 151
select = j
return
end

c *** performs single-point crossover
c     between mate1 and mate2 ***
subroutine crossover(ipop,mate1,mate2,oldchr,
-           newchr)
common/sgaparam/ipopsiz,1chrom,maxgen,ncross,
-           nmute,pcross,pmute,nparam

```

```

integer*1 oldchr(ipopsizelchrom)
integer*1 newchr(ipopsizelchrom)
c.....check if a crossover is to be performed
if(iflip(pcross) .eq. 1) then
c.....if yes, create a random cross site
    jcross = irnd(1,lchrom-1)
    ncross = ncross + 1
else jcross = lchrom
endif
c.....copy till the cross site as it is
do 171 j = 1,jcross
    newchr(ipop, j) = mutation(oldchr(mate1,j))
    newchr(ipop+1,j) = mutation(oldchr(mate2,j))
171 continue
if(jcross .eq. lchrom) go to 173
c.....swap from the cross site till the end of string
do 172 j = jcross + 1,lchrom
    newchr(ipop, j) = mutation(oldchr(mate2,j))
    newchr(ipop+1,j) = mutation(oldchr(mate1,j))
172 continue
173 return
end

c     **** returns 1 if ivalue = 0 and vice versa ****
function logicalnot(ivalue)
integer*1 ivalue
logicalnot = 0
if(ivalue .eq. 0) logicalnot = 1
return
end

c     *** mutates ivalue with probability 'pmute' ***
function mutation(ivalue)
integer*1 ivalue
common/sgaparam/ipopsizelchrom,maxgen,ncross,
-          nmute,pcross,pmute,nparam
mutation = ivalue
c.....check if a mutation is to be performed
if(iflip(pmute) .eq. 1) then
c..... if yes, perform mutation
    mutation = logicalnot(ivalue)
    nmute = nmute + 1
endif
return
end

```

```

c      ***** prints report of a generation *****
subroutine report(olidx,newx,oldfit,newfit,bestsol)
common/sgaparam/ipopsiz,lpchrom,maxgen,ncross,
-          nmute,pcross,pmute,npam
common/statist/igen,avg,amax,amin,sumfitness
common/bestpt/ibest,bestfit,igenbst
real olidx(ipopsiz,npam), newx(ipopsiz,npam)
real oldfit(ipopsiz), newfit(ipopsiz)
real bestsol(npam)
write (*,9150) igen
write (*,*) ' '
write (*,9151)
write (*,*) ' '
c.....report the old population
do 131 j = 1,ipopsiz
    write(*,201) j
    do 132 i = 1,npam
132        write(*,202) olidx(j,i)
        write(*,203) oldfit(j)
        write(*,205)
c.....report the new population
        do 133 i = 1,npam
133        write(*,202) newx(j,i)
        write(*,204) newfit(j)
131 continue
c.....update the best solution by the new best solution
    if (newfit(ibest) .gt. bestfit) then
        bestfit = newfit(ibest)
        igenbst = igen
        do 134 j = 1,npam
134        bestsol(j) = newx(ibest,j)
    endif
    write (*,*) ' '
c.....print the best solution
    write (*,206)
    do 135 j = 1,npam
135        write (*,202) bestsol(j)
        write (*,9159) bestfit,igenbst
201 format(1x,i3,'$',f7.3,$)
202 format(f7.3,$)
203 format(f9.5,$)
204 format(f9.5)
205 format(' | ',f9.5,$)
206 format(1x,'Best solution so far --> ',f10.5,
9159 format(/,1x,'Fitness = ',f10.5,
-          ' found at generation = ',i4,/)
```

```

c.....print the populaion statistics
    write (*,*)
    - 'Population statistics at this generation'
    write (*,9153) amax
    write (*,9154) avg
    write (*,9155) amin
    write (*,9156) ncross
    write (*,9157) nmutate
    write (*,*) ''
    return
9150 format(1x,'Accumulated statistics for generation',
            '-      '-->  ',i4)
9151 format(1x,'           x      fitness |',
            '-           ,       x      fitness')
9153 format(1x,'maximum fitness      =  ',f10.5)
9154 format(1x,'average fitness      =  ',f10.5)
9155 format(1x,'minimum fitness      =  ',f10.5)
9156 format(1x,'number of crossovers =  ',i6)
9157 format(1x,'number of mutations   =  ',i6)
    end

c ***** decodes a substring of length lstr *****
function decode(chrom,id,lstr)
common/sgaparam/ipopsizelchrom,maxgen,ncross,
-          nmutate,pcross,pmute,npParam
    integer*1 chrom(lchrom)
    acc = 0.0
c.....multiplier is one for the first bit position
    powof2 = 1.0
    do 71 j = 1,lstr
c..... accumulate the bit value times the multiplier
        if(chrom(id+j) .eq. 1) acc = acc + powof2
c..... update multiplier for next bit position
        powof2 = powof2 * 2
71    continue
    decode = acc
    return
end

c * map the decoded value in the specified region *
function xmap(y,alo,ahi,denom)
c.....linear mapping
    xmap = alo + (ahi-alo)*y/denom
    return
end

```

```

c      ** calculate the parameter value from a string **
subroutine decodevars(chrom,x,lsubstr,alow,ahigh,
-                      factor)
common/sgaparam/ipopsizelchrom,maxgen,ncross,
-                      nmute,pcross,pmute,npnparam
integer*1 chrom(lchrom)
integer lsubstr(npnparam)
real alow(npnparam),ahigh(npnparam),factor(npnparam)
real x(npnparam)
jpos = 0
do 10 j = 1,npnparam
c.....get the decoded value of the substring
y = decode(chrom,jpos,lsubstr(j))
c..... map in the specified range for real value
x(j) = xmap(y,alow(j),ahigh(j),factor(j))
c..... bit position for the next variable in the string
jpos = jpos + lsubstr(j)
10 continue
return
end

c      ***** copy new population to old population *****
subroutine copynewtoold(oldchr,newchr,oldx,newx,
-                      oldfit,newfit)
common/sgaparam/ipopsizelchrom,maxgen,ncross,
-                      nmute,pcross,pmute,npnparam
integer*1 oldchr(ipopsizelchrom)
integer*1 newchr(ipopsizelchrom)
real oldx(ipopsizelnpnparam), newx(ipopsizelnpnparam)
real oldfit(ipopsizel), newfit(ipopsizel)
do 191 i = 1,ipopsizel
c..... copy the strings
do 192 j = 1,lchrom
oldchr(i,j) = newchr(i,j)
192 continue
c..... copy the variables
do 193 j = 1,npnparam
oldx(i,j) = newx(i,j)
193 continue
c..... finally, copy the fitness
oldfit(i) = newfit(i)
191 continue
return
end

```

```

c..... utility subroutines .....
c      ***** function to find i mod j *****
function modop(i,j)
modop = i - j*ifix(float(i)/float(j))
return
end

c *** random number initialization routines ***
subroutine advance_random
common /randvar/ oldrand,jrand
real oldrand(55)
do 11 j1 = 1,24
    rand_new = oldrand(j1) - oldrand(j1+31)
    if(rand_new.lt.0.0) rand_new = rand_new + 1.0
    oldrand(j1) = rand_new
11 continue
do 12 j1 = 25,55
    rand_new = oldrand(j1)-oldrand(j1-24)
    if(rand_new.lt.0.0) rand_new = rand_new+1.0
    oldrand(j1) = rand_new
12 continue
return
end

subroutine warmup_random(random_seed)
real oldrand(55)
common/randvar/oldrand,jrand
oldrand(55) = random_seed
rand_new = 1.0e-9
prev_rand = random_seed
do 21 j1 = 1,54
    ii = modop(21*j1,55)
    oldrand(ii) = rand_new
    rand_new = prev_rand - rand_new
    if(rand_new.lt.0.0) rand_new = rand_new + 1.0
    prev_rand = oldrand(ii)
21 continue
call advance_random
call advance_random
call advance_random
jrand = 0
return
end

c ***** create a random number between 0 and 1 *****
function random()

```

```

real oldrand(55)
common/randvar/oldrand,jrand
jrand = jrand + 1
if(jrand.gt.55) then
    jrand = 1
    call advance_random
endif
random = oldrand(jrand)
return
end

c ***** flip a coin with a probability prob *****
function iflip(prob)
iflip = 0
if ((prob.eq.1.0).or.(random().le.prob)) iflip = 1
return
end

c * create a random integer between ilow and ihigh *
function irnd(ilow,ihigh)
if(ilow.ge.ihigh) then
    i = ilow
else
    i = ifix(random()*(ihigh-ilow+1)+ilow)
    if(i.gt.ihigh) i = ihigh
endif
irnd = i
return
end

c ***** initiate the random number generator *****
subroutine randomize
common/consts/fix
61 write(*,*)'enter seed random number (0.0 to 1.0)>'
read (*,*) randomseed
if ((randomseed.lt.0) .or. (randomseed.gt.1.0))
- go to 61
call warmup_random(randomseed)
fix = randomseed
return
end

```

## Simulation Run

The above code runs successfully on a PC-386 under Microsoft FORTRAN. The two-variable unconstrained Himmelblau function is coded in the function funct for minimization. In order to demonstrate the working of the code, we present the results of one simulation run. The input values to the code are given below.

```

Genetic Algorithms in FORTRAN
Kalyanmoy Deb
All rights reserved

** GA data entry and initialization **
enter number of parameters ----->
2
enter total string length ----->
20
enter population size ----->
20
enter max. generation ----->
30
enter crossover probability ----->
0.8
enter mutation probability ----->
0.05
enter string length for variable 1
10
enter minimum and maximum bound on variable 1
0 5
enter string length for variable 2
10
enter minimum and maximum bound on variable 2
0 5
enter seed random number (0.0 to 1.0)>
0.1234

```

The termination criterion is set to perform 30 generations of GA simulation. Three generations of GA simulation are presented in the following. In each generation, the old population and the new population are printed.

```

Genetic Algorithms in FORTRAN
Kalyanmoy Deb
All rights reserved

GA parameters
-----
population size      =  20

```

```

chromosome length      =   20
max. # of generations =   30
crossover probability =  0.800
mutation probabiltiy =  0.050
seed random number    =   0.123

maximum fitness        =   0.10435
average fitness         =   0.02369
minimum fitness         =   0.00243

```

Accumulated statistics for generation--> 1

	x	fitness		x	fitness		
1)	1.931	0.318	0.01350		0.860	4.326	0.00515
2)	0.836	4.301	0.00529		0.279	3.964	0.00768
3)	2.678	0.621	0.03736		2.165	1.955	0.04760
4)	2.791	2.869	0.05782		2.630	2.869	0.05752
5)	2.776	1.408	0.10435		0.836	4.301	0.00529
6)	0.279	4.013	0.00731		2.791	3.025	0.03925
7)	0.235	3.964	0.00771		2.620	2.131	0.19803
8)	0.440	4.115	0.00652		0.547	1.408	0.00932
9)	1.799	2.131	0.03014		2.678	1.481	0.09066
10)	2.292	4.027	0.00733		4.027	0.547	0.02415
11)	2.004	1.960	0.03626		0.440	1.926	0.01136
12)	1.931	3.089	0.02597		1.799	4.164	0.00620
13)	2.942	3.250	0.02265		2.688	3.123	0.03221
14)	4.966	3.568	0.00243		2.287	4.027	0.00733
15)	1.637	4.130	0.00644		3.871	1.496	0.03142
16)	2.937	3.822	0.00875		2.620	1.408	0.07040
17)	3.436	2.424	0.05958		3.407	1.955	0.13145
18)	0.973	4.091	0.00661		2.004	1.725	0.03055
19)	0.890	1.799	0.01250		2.786	0.367	0.03862
20)	1.139	3.270	0.01522		2.107	1.569	0.03140

Best solution so far --> 2.620 2.131  
Fitness = 0.19803 found at generation = 1

Population statistics at this generation  
maximum fitness = 0.19803  
average fitness = 0.04378  
minimum fitness = 0.00515  
number of crossovers = 5  
number of mutations = 24

Accumulated statistics for generation--> 2

	x	fitness		x	fitness
1)	0.860	4.326	0.00515		2.620 2.111 0.19574
2)	0.279	3.964	0.00768		2.439 1.955 0.08980
3)	2.165	1.955	0.04760		2.791 2.170 0.42838
4)	2.630	2.869	0.05752		0.850 4.301 0.00528
5)	0.836	4.301	0.00529		2.542 3.069 0.03598
6)	2.791	3.025	0.03925		2.791 2.107 0.43999
7)	2.620	2.131	0.19803		0.860 1.823 0.01248
8)	0.547	1.408	0.00932		2.678 3.983 0.00744
9)	2.678	1.481	0.09066		3.050 1.559 0.30053
10)	4.027	0.547	0.02415		0.802 1.408 0.01011
11)	0.440	1.926	0.01136		2.620 2.869 0.05735
12)	1.799	4.164	0.00620		2.551 2.131 0.15204
13)	2.688	3.123	0.03221		2.165 2.072 0.05191
14)	2.287	4.027	0.00733		2.678 1.911 0.19546
15)	3.871	1.496	0.03142		1.804 4.164 0.00620
16)	2.620	1.408	0.07040		2.620 1.505 0.08068
17)	3.407	1.955	0.13145		0.283 3.964 0.00767
18)	2.004	1.725	0.03055		2.625 2.869 0.05743
19)	2.786	0.367	0.03862		0.127 0.469 0.00642
20)	2.107	1.569	0.03140		2.776 2.004 0.36967

Best solution so far --> 2.791 2.107  
 Fitness = 0.43999 found at generation = 2

Population statistics at this generation  
 maximum fitness = 0.43999  
 average fitness = 0.12553  
 minimum fitness = 0.00528  
 number of crossovers = 12  
 number of mutations = 44

Accumulated statistics for generation--> 3

	x	fitness		x	fitness
1)	2.620	2.111	0.19574		4.042 2.483 0.01380
2)	2.439	1.955	0.08980		2.791 1.481 0.12409
3)	2.791	2.170	0.42838		2.620 2.107 0.19509
4)	0.850	4.301	0.00528		2.791 2.116 0.44017
5)	2.542	3.069	0.03598		2.776 1.544 0.13446
6)	2.791	2.107	0.43999		3.060 1.559 0.30443
7)	0.860	1.823	0.01248		2.776 2.087 0.40457
8)	2.678	3.983	0.00744		3.690 4.042 0.00465

9)	3.050	1.559	0.30053		2.713	3.069	0.03612
10)	0.802	1.408	0.01011		2.542	2.180	0.15017
11)	2.620	2.869	0.05735		3.871	2.087	0.02540
12)	2.551	2.131	0.15204		2.752	2.884	0.05606
13)	2.165	2.072	0.05191		2.620	1.486	0.07844
14)	2.678	1.911	0.19546		0.782	2.033	0.01331
15)	1.804	4.164	0.00620		2.361	1.960	0.07322
16)	2.620	1.505	0.08068		0.860	1.823	0.01248
17)	0.283	3.964	0.00767		2.674	2.732	0.08262
18)	2.625	2.869	0.05743		2.796	1.237	0.08368
19)	0.127	0.469	0.00642		3.196	2.498	0.10135
20)	2.776	2.004	0.36967		0.547	4.301	0.00535

Best solution so far --> 2.791 2.116  
Fitness = 0.44017 found at generation = 3

Population statistics at this generation  
maximum fitness = 0.44017  
average fitness = 0.11697  
minimum fitness = 0.00465  
number of crossovers = 20  
number of mutations = 69

Finally, we present the statistics of the 30th generation. The results show that the population converges near the true minimum point  $(3, 2)^T$ . Note that the obtained best solution  $(2.962, 2.033)^T$  gets generated at generation 28.

Accumulated statistics for generation--> 30

	x	fitness		x	fitness		
1)	3.099	1.114	0.12265		2.913	1.906	0.63464
2)	3.705	2.175	0.03694		2.913	1.945	0.70635
3)	2.903	0.846	0.06487		2.913	0.899	0.06966
4)	2.923	0.684	0.05824		2.654	0.694	0.03763
5)	2.967	0.665	0.06154		2.962	1.877	0.72011
6)	2.947	2.258	0.47678		2.962	1.877	0.72011
7)	2.962	1.877	0.72011		2.654	1.095	0.05237
8)	2.967	2.151	0.73951		2.776	0.684	0.04564
9)	0.606	4.848	0.00305		2.815	2.468	0.19978
10)	2.913	1.945	0.70635		3.705	3.661	0.00695
11)	2.908	1.935	0.67182		2.908	2.092	0.77875
12)	2.962	2.072	0.91762		0.406	1.935	0.01132
13)	2.654	2.033	0.21080		2.933	2.258	0.47985
14)	4.213	2.072	0.01233		1.862	4.848	0.00289
15)	3.710	2.151	0.03716		2.967	4.653	0.00315
16)	2.654	3.324	0.02160		2.942	2.190	0.63842

17)	4.370	1.921	0.00977		2.908	2.263	0.46787
18)	2.967	4.536	0.00357		0.445	1.935	0.01143
19)	2.903	2.033	0.77466		3.079	2.151	0.52774
20)	2.962	1.310	0.13822		2.967	0.684	0.06249

Best solution so far --> 2.962 2.033

Fitness = 0.95524 found at generation = 28

Population statistics at this generation

maximum fitness = 0.77875

average fitness = 0.30886

minimum fitness = 0.00289

number of crossovers = 233

number of mutations = 654

# Appendix

## Linear Programming Algorithms

---

Linear programming methods solve optimization problems having linear objective function and constraints. That is why these methods are used extensively in operations research or transportation problems. However, linear programming methods are also used iteratively to solve nonlinear programming problems where the objective function and/or constraints are nonlinear. In Chapter 4, we have presented at least two algorithms (the Frank-Wolfe method and the cutting plane method) which iteratively use the linear programming method. In this Appendix, we give a brief description of the simplex method of the linear programming technique. Some numerical examples are illustrated to make the understanding of the algorithm clear. For readers who have had some exposure to linear programming methods, this discussion will be a refresher. For readers who have not had any exposure of linear programming methods earlier, this chapter may hopefully generate interest to refer to more detailed books on linear programming methods (Dantzig, 1963; Murty, 1983; Taha, 1989).

### A.1 Linear Programming Problem

Linear programming methods can solve problems having linear objective function and linear constraints. Since all constraints are linear, the feasible region is usually an enclosed region surrounded by linear hyperplanes. Since the objective function is also linear, the optimum point is usually one of the corner points (or hyperplanes) of the feasible region. Here we consider a maximization problem and discuss two ways to handle minimization problems later. Let us consider an  $N$ -variable problem having  $J$  equality constraints suitable for an LP method:

$$\begin{aligned}
 & \text{Maximize} && f(x) = c_1x_1 + c_2x_2 + \dots + c_Nx_N \\
 & \text{subject to} && \\
 & && \left. \begin{array}{l}
 a_{11}x_1 + a_{12}x_2 + \dots + a_{1N}x_N = b_1, \\
 a_{21}x_1 + a_{22}x_2 + \dots + a_{2N}x_N = b_2, \\
 \vdots \\
 a_{J1}x_1 + a_{J2}x_2 + \dots + a_{JN}x_N = b_J, \\
 x_i \geq 0, \quad i = 1, 2, \dots, N.
 \end{array} \right\} \quad (\text{A.1})
 \end{aligned}$$

Even though all constraints in the above problem are shown to be equality constraints, inequality constraints can also be handled using the linear programming method. An inequality constraint is usually transformed into an equivalent equality constraint by introducing a slack variable. If the inequality constraint is of lesser-than-equal-to type, a slack variable is added to the left-side expression. For example, the constraint  $g_j(x) \leq 0$  can be converted into an equality constraint by adding a slack variable  $x_{N+j}$  as follows:

$$g_j(x) + x_{N+j} = 0,$$

where  $x_{N+j} \geq 0$ . The idea is that since the quantity  $g_j(x)$  is less than or equal to zero, we add a positive quantity to make the sum zero. On the other hand, if the constraint is of greater-than-equal-to type, a slack variable is subtracted from the left side expression. The constraint  $g_j(x) \geq 0$  is converted to

$$g_j(x) - x_{N+j} = 0,$$

where  $x_{N+j} \geq 0$ . The final set of constraints in Equation (A.1) requires all variables to take nonnegative values. If any variable is expected to have a negative value, that variable can be substituted by another variable (which can take only nonnegative values) using a suitable linear transformation. In the above transformations of inequality to equality constraints, the slack variables are fixed to take nonnegative values. Thus, any linear problem having inequality or equality constraints can be rewritten in the form given in Problem A.1.

In general, in a real-world optimization problem, there are more variables than the number of constraints. In those situations, equality constraints can be used to express  $J$  variables in terms of other  $(N-J)$  variables. The equality constraints can be written in the *row-echelon* form as follows (Strang, 1980):

$$\begin{aligned}
 x_1 &+ a'_{1(J+1)}x_{J+1} + \dots + a'_{1N}x_N = b'_1, \\
 x_2 &+ a'_{2(J+1)}x_{J+1} + \dots + a'_{2N}x_N = b'_2, \\
 \ddots &\quad \vdots \quad \vdots \quad \vdots \quad \vdots \\
 x_J &+ a'_{J(J+1)}x_{J+1} + \dots + a'_{JN}x_N = b'_J. \quad \left. \right\} \quad (\text{A.2})
 \end{aligned}$$

The variables  $x_1$  to  $x_J$  are expressed in terms of other variables ( $x_{J+1}$  to  $x_N$ ) and are known as *basic* variables. The variables  $x_{J+1}$  to  $x_N$  can take any values and are known as *nonbasic* variables. It is important to note that even though the above equations show that the first  $J$  variables ( $x_1$  to  $x_J$ ) are basic variables, in practice, any  $J$  variables can be considered as basic variables. In fact, basic variables are chosen using some guidelines, which we shall discuss later. In the above row-echelon formulation, the coefficients corresponding to the basic variables are always one and  $b'_j$  values are always positive. For a set of values of the nonbasic variables, a feasible solution to the above problem can be obtained by computing other  $J$  variables. Therefore, there are infinite feasible solutions to the above problem satisfying all the above equations. Since the objective in problem (A.1) is to maximize the function  $f(x)$ , the feasible solution(s) corresponding to the maximum function value is(are) the optimum solution(s) to the problem.

There are two different ways one can get a feasible solution of the original problem. One way to achieve this is to choose the set of basic variables a priori, assign several sets of values to nonbasic variables, and then obtain corresponding basic variables in each case using (A.2). The other way is to choose several sets of basic variables, assign all nonbasic variables to zero, and then compute the values of the basic variables in each case using (A.2). In the method described in the following section, the latter approach is used. In this approach, any combination of  $J$  variables can be chosen to create the row-echelon formulation. Since there are  $N$  variables from which to choose  $J$  variables, there are a total of  $\binom{N}{J}$  different solutions to compare<sup>1</sup>. A basic feasible solution is defined as a solution at which all basic variables are nonnegative and all nonbasic variables are zero. Thus, there are a total of  $\binom{N}{J}$  basic feasible solutions possible, of which one or more corresponds to the optimum point. A naive approach would be to compare all such basic feasible solutions and obtain the optimum. An efficient approach would be to compare only a small fraction of all basic feasible solutions to obtain the optimum. In the following section, we discuss one such efficient algorithm.

## A.2 Simplex Method

The simplex method was developed by G. B. Dantzig in 1963. The main concept of the simplex method is to compare *neighbouring* basic feasible solutions in an efficient way. A neighbouring solution is a basic feasible solution which differs from the current basic feasible solution in exactly one basic variable. That is, one basic variable in the current basic feasible solution is made nonbasic and a currently nonbasic variable is made basic. This can be achieved in a total of  $J(N - J)$  different ways. Therefore, there are two decisions to be made:

---

<sup>1</sup>The combination  $\binom{N}{J}$  is calculated by computing the factorials of  $N$ ,  $J$ , and  $(N - J)$  as follows:  $\binom{N}{J} = N!/(J!(N - J)!)$ , where  $N! = N(N - 1)(N - 2) \cdots 2 \times 1$ .

- (i) Which one of the  $(N - J)$  nonbasic variables is to be chosen for the basic variable?
- (ii) Which one of the  $J$  basic variables is to be chosen for the nonbasic variable?

We answer these questions now.

It is clear that the choice of different nonbasic variables will result in different objective function values. The simplex method chooses that nonbasic variable which causes a maximum increase in the objective function value. By simple algebraic calculations, it can be shown that the increase in the objective function value  $f(x)$  due to a unit increase in a nonbasic variable  $x_q$  from zero to one is given by

$$(\Delta f)_q = c_q - \sum_{j=1}^J c_j a'_{jq}.$$

Thus, we can calculate  $(\Delta f)_q$  for all nonbasic variables and choose the one with the maximum positive value. In the case of a tie (same  $(\Delta f)_q$  for more than one nonbasic variables), any nonbasic variable can be chosen at random. It is worth mentioning here that if for all remaining nonbasic variables, the quantity  $(\Delta f)_q$  is non-positive, no increment in the objective function value is possible. This suggests that the optimum solution has been obtained, and we terminate the algorithm.

Once a nonbasic variable ( $x_q$ ) is chosen, the next question is, which of the basic variables has to be made nonbasic. It is clear that as  $x_q$  is increased from zero to one, the objective function value will also increase. Therefore, we may want to increase the nonbasic variable  $x_q$  indefinitely. But there is a limit to the extent of this increment. When  $x_q$  is increased, all basic variables must either be increased, decreased, or kept the same in order to make the solution feasible. Recall that all variables in a linear program must be nonnegative. Thus, the critical basic variable is the one which, when reduced, becomes zero first. Any more increase in the chosen nonbasic variable will make that basic variable negative. From the row-echelon formulation, we can write the value of a basic variable as follows:

$$x_j = b'_j - a'_{jq} x_q, \quad j = 1, 2, \dots, J.$$

A basic variable becomes zero when  $x_q = b'_j/a'_{jq}$ . Since, in the row-echelon form, all  $b'_j$  are nonnegative, this can happen only when  $a'_{jq}$  is positive. In order to find the critical basic variable, we compute the quantity  $b'_j/a'_{jq}$  and choose the basic variable  $x_q$  for which this quantity is minimum. This rule is also known as *minimum ratio* rule in linear programming.

The simplex method begins with an initial feasible solution. Thereafter, a basic variable is replaced by a nonbasic variable chosen according to rules described above. Thus, the simplex method is an iterative method which works by alternating among various basic and nonbasic variables so as to achieve the optimum point efficiently. Since all constraints and the objective

function are linear, these points are the corners of the feasible search region. In the following, we describe the algorithm:

### Algorithm

**Step 1** Choose a basic feasible solution. Set all nonbasic variables to zero.

**Step 2** Calculate the quantity  $(\Delta f)_q$  for all nonbasic variables and choose the one having the maximum value. If  $(\Delta f)_q \leq 0$  for all nonbasic variables, **Terminate**;

Else use the minimum ratio rule to choose the basic variable to be replaced.

**Step 3** Perform a row-echelon formulation for new basic variables and go to Step 2.

The algorithm assumes that the underlying problem can be written in the form shown in Equation (A.1). If some variables in the original problem take negative values, they need to be suitably transformed so that the new variables can only take nonnegative values. Inequality constraints are converted into equality constraints by adding or subtracting slack variables. The algorithm stated above works only for maximization problems. There are two ways the above algorithm can be used to solve minimization problems. The duality principle (by multiplying the objective function by  $-1$ ) can be used to convert the minimization problem into an equivalent maximization problem. In the other approach, the criterion for choosing a nonbasic variable to make it basic needs to be changed. Recall that in the maximization problem, the nonbasic variable for which the quantity  $(\Delta f)_q$  is maximally nonnegative is chosen. That variable increases the objective function at the highest rate. Similarly, the nonbasic variable for which the quantity  $(\Delta f)_q$  is maximally non-positive will decrease the objective function value at the highest rate. Thus, the above algorithm can be used for the minimization problems by selecting nonbasic variables according to the most negative  $(\Delta f)_q$ . The rest of the algorithm can be used as above. The other change is that the algorithm is terminated when all nonbasic variables have nonnegative values of the quantity  $(\Delta f)_q$ .

We illustrate the working of the simplex algorithm by considering a simple constrained optimization problem.

### EXERCISE A.2.1

Consider the following problem:

$$\text{Maximize } f(\mathbf{x}) = 2x_1 + 3x_2,$$

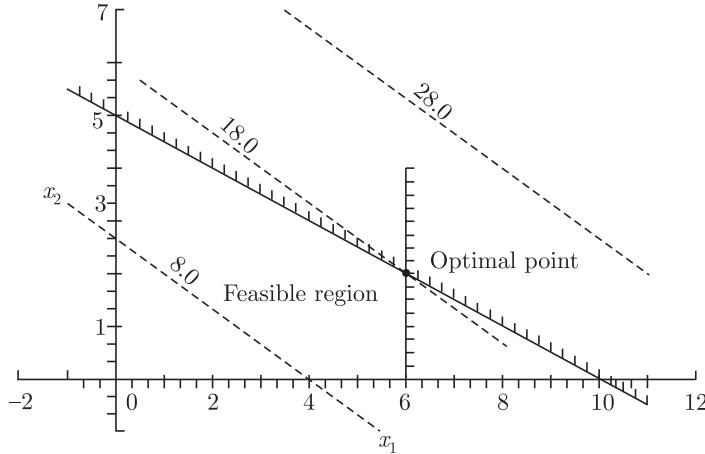
subject to

$$x_1 \leq 6,$$

$$x_1 + 2x_2 \leq 10,$$

$$x_1, x_2 \geq 0. \quad (\text{A.3})$$

The feasible solution space and the optimum solution are shown in Figure A.1. It is clear that the optimum solution is  $(6, 2)^T$  with a function value equal to 18. Let us investigate how we can obtain this solution using the simplex method.



**Figure A.1** The linear program described in Exercise A.2.1. The feasible region and the optimal solution are shown.

**Step 1** First of all, we have to convert each inequality constraint into an equality constraint using a slack variable. Two inequality constraints are converted into the two equality constraints as follows:

$$\begin{aligned} x_1 &+ x_3 = 6, \\ x_1 + 2x_2 &+ x_4 = 10. \end{aligned}$$

The variables  $x_3$  and  $x_4$  are slack variables and can take only nonnegative values. Thus, the optimization problem becomes as follows:

$$\text{Maximize } f(x) = 2x_1 + 3x_2$$

subject to

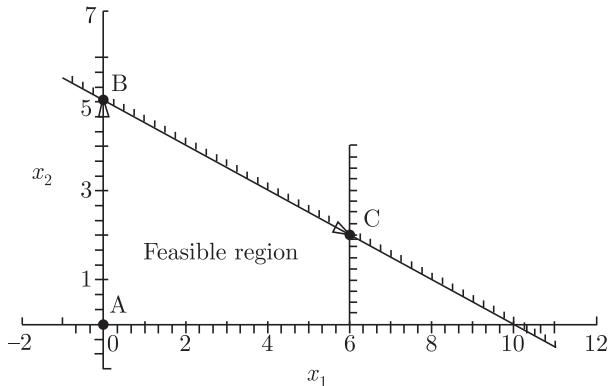
$$\begin{aligned} x_1 &+ x_3 = 6, \\ x_1 + 2x_2 &+ x_4 = 10, \\ x_1, x_2, x_3, x_4 &\geq 0. \end{aligned}$$

This problem is now well represented by the problem described in Equation (A.1). We also observe that both constraints are already in row-echelon form with  $x_3$  and  $x_4$  being basic variables. Thus, we begin the simplex method with the initial feasible solution:  $x_1 = 0$ ,  $x_2 = 0$ ,  $x_3 = 6$ , and  $x_4 = 10$ . The constraints can be represented in the following tableau:

		2	3	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
0	$x_3$	1	0	1	0	6 $\frac{6}{0} = \infty$
0	$x_4$	1	2	0	1	10 $\frac{10}{2} = 5 \leftarrow$
$(\Delta f)_q$		2	3	0	0	$f(x) = 0$

↑

The top row of the table represents the coefficients of the variables in the objective function. The next row shows all variables. The next two rows show the coefficients ( $a_{ij}$ ) in the constraint expressions. There will be as many rows as there are constraints. The seventh column shows the corresponding  $b_j$  values in the constraints. The second column shows the basic variables and the first column shows the corresponding coefficients in the objective function. The bottom row shows the quantity  $(\Delta f)_q$  for all variables. For basic variables, this quantity must be zero. For nonbasic variables, this quantity is calculated in Step 2. The above table represents the current linear programming problem and it contains all the information necessary to form the next row-echelon table. The table also represents the current solution—all nonbasic variables are zero and basic variables ( $x_q$ ) are set equal to  $b_j$  where the row  $j$  corresponds to the coefficient  $a_{qj} = 1$  of the basic variable. It is important to note that in the column of every basic variable there must be only one entry of  $a_{ij} = 1$  and rest of the entries must be zero. This initial solution is marked as A in Figure A.2.



**Figure A.2** The intermediate points found in the simulation of the simplex algorithm. The figure shows how neighbouring basic feasible solutions are found iteratively and the optimal solution is finally obtained.

**Step 2** The quantities  $(\Delta f)_q$  are calculated for the nonbasic variables and are shown in the bottom row. It turns out that the quantity  $(\Delta f)_q$  is maximum for the variable  $x_2$ . Thus, the nonbasic variable  $x_2$  is to be made a basic variable. Since  $(\Delta f)_2$  is positive, the optimal solution is not found. We use

the minimum ratio rule to determine the basic variable which is to be replaced. The right-most column shows these calculations. By calculating the quantity  $b_j/a_{2j}$ , we observe that the quantity is minimum for the basic variable  $x_4$ . Thus, the new basic variables are  $x_2$  and  $x_3$ , and the new nonbasic variables are  $x_1$  and  $x_4$ .

**Step 3** We create a new row-echelon table using variables  $x_2$  and  $x_3$  as basic variables. The new parameter values are shown in the following tableau:

		2	3	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
0	$x_3$	[1]	0	1	0.0	6 $\frac{6}{1} = 6 \leftarrow$
3	$x_2$	0.5	1	0	0.5	5 $\frac{5}{0.5} = 10$
$(\Delta f)_q$		0.5	0	0	-1.5	$f(x) = 15$

↑

The above table represents the solution  $x_1 = 0$ ,  $x_2 = 5$ ,  $x_3 = 6$  and  $x_4 = 0$ . This solution is marked as B in Figure A.2. Notice that in one iteration, the simplex method moves to a neighbouring point of the initial point.

**Step 2** Next, the quantity  $(\Delta f)_q$  is calculated for all nonbasic variables. It is observed that only  $(\Delta f)_1$  is positive. Thus, the variable  $x_1$  is chosen to be the new basic variable. In order to decide which of the two older basic variables to replace, we use the minimum ratio rule. It turns out that the variable  $x_3$  has the smallest  $b_j/a_{1j}$  value. Thus, the new basic variables are  $x_1$  and  $x_2$ .

**Step 3** At this step, we create a new row-echelon matrix with  $x_1$  and  $x_2$  as basic variables and obtain the following tableau:

		2	3	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
2	$x_1$	1	0	1.0	0.0	6
3	$x_2$	0	1	-0.5	0.5	2
$(\Delta f)_q$		0	0	-0.5	-1.5	$f(x) = 18$

The solution at this step is  $x_1 = 6$ ,  $x_2 = 2$ ,  $x_3 = 0$ , and  $x_4 = 0$ . This solution is marked as C in Figure A.2.

**Step 2** At this step, the quantity  $(\Delta f)_q$  for all nonbasic variables is non-positive. Thus, the optimal solution is found. The optimal solution is  $x^* = (6, 2)^T$  and the optimal function value is  $f(x^*) = 18$ .

Figure A.2 shows that the simplex method begins at a corner point of the feasible search region and visits only the neighbouring corner points in successive iterations. Since the optimal solution is bound to be one of the corner points of the feasible bounded region, the simplex method guarantees the convergence to the optimal solution in linear programming problems.

In the case of an unbounded optimal solution (where the optimal function value is not finite), the constraint entries for all basic variables corresponding to the chosen nonbasic variables are negative.

### A.3 Artificial Variables and Dual Phase Method

The working principle of the simplex method is straightforward and is easy to implement. There exist some commercial codes implementing the simplex method (IBM MPSX-370 or Management Science Systems MPS-III). However, a successful LP code must handle a difficulty usually encountered in many LP problems. The difficulty is that in some LP problems it may not be possible to obtain an initial basic feasible solution directly from the given constraints. In those situations, *artificial* variables are added. Artificial variables have no meaning as far as the given problem is concerned. They are simply included to obtain a basic feasible solution. Once a basic feasible solution is found, the artificial variables can be excluded from further consideration. Usually, a dual phase strategy is used for this purpose. In the first phase, the simplex method is used to find a basic feasible solution comprising the design variables, slack variables, and artificial variables (if any). The objective of the search strategy is to maximize the negative of the sum of the artificial variables. Since the artificial variables are also constrained to be nonnegative, the optimal solution of the first phase is a solution at which all artificial variables are zero. Since at the end of the first phase all artificial variables are zero, the basic variables are either design variables or slack variables. This constitutes a basic feasible solution which can be used as the initial solution for the next phase, where  $f(x)$  is used as the objective function. We illustrate the dual phase method by solving the following exercise problem.

#### **EXERCISE A.3.1**

Let us add one more inequality constraint to the problem used in the previous exercise. Thus, the new problem is as follows:

$$\text{Maximize } f(x) = 2x_1 + 3x_2$$

subject to

$$\begin{aligned} x_1 &\leq 6, \\ x_1 + 2x_2 &\leq 10, \\ x_1 + x_2 &\geq 2, \\ x_1, x_2 &\geq 0. \end{aligned}$$

The solution to the above problem remains the same as that in the previous problem:  $x^* = (6, 2)^T$ .

**Step 1** In order to convert the new inequality constraint into an equality constraint, we use another slack variable:

$$x_1 + x_2 - x_5 = 2,$$

where  $x_5$  is nonnegative. We write all three constraints in the row-echelon form:

$$\begin{aligned} x_1 &+ x_3 &= 6, \\ x_1 + 2x_2 &+ x_4 &= 10, \\ x_1 + x_2 &- x_5 &= 2, \\ x_1, x_2, x_3, x_4, x_5 &\geq 0. \end{aligned}$$

Since there are three constraints, there must be three basic variables. The variables  $x_3$  and  $x_4$  can be considered as basic variables. But the variables  $x_1$ ,  $x_2$ , and  $x_5$  cannot be considered as basic variables, because none of them have a coefficient one in one equation and zero in other equations. Thus, we add an artificial variable  $x_6$  at the third constraint and form the following row-echelon form:

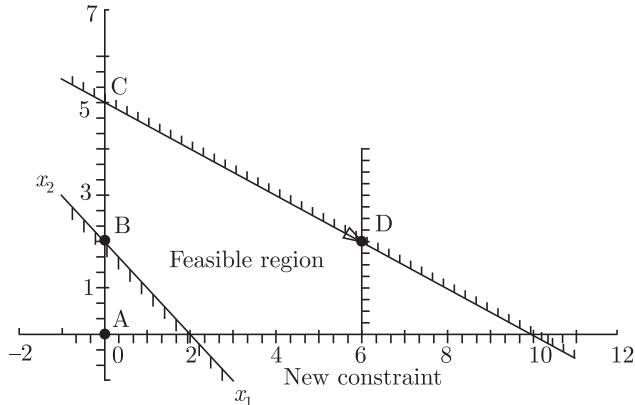
$$\begin{aligned} x_1 &+ x_3 &= 6, \\ x_1 + 2x_2 &+ x_4 &= 10, \\ x_1 + x_2 &- x_5 + x_6 &= 2, \\ x_1, x_2, x_3, x_4, x_5, x_6 &\geq 0. \end{aligned}$$

Now, the variables  $x_3$ ,  $x_4$ , and  $x_6$  can be considered as basic variables. Thus, variables  $x_1$ ,  $x_2$ , and  $x_5$  are nonbasic variables. The initial basic feasible solution is  $x_1 = x_2 = x_5 = 0$ ,  $x_3 = 6$ ,  $x_4 = 10$ , and  $x_6 = 2$ . This is the initial solution of the first phase where the objective function is  $-x_6$ . We form the following tableau:

		0	0	0	0	0	-1	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
0	$x_3$	1	0	1	0	0	0	$\frac{6}{0} = \infty$
0	$x_4$	1	2	0	1	0	0	$\frac{10}{2} = 5$
-1	$x_6$	1	<span style="border: 1px solid black; padding: 2px;">1</span>	0	0	-1	1	$\frac{2}{1} = 2 \leftarrow$
$(\Delta f)_q$		1	1	0	0	-1	0	$f(x) = -2$

↑

The initial solution is  $(0, 0)^T$  and is marked as A in Figure A.3. It is interesting to note that this solution is not a feasible solution to the original problem, violating the third constraint.



**Figure A.3** The intermediate points found in the simulation of the dual phase method of the simplex algorithm. The first phase begins with the point A and the basic feasible solution B is found. In the next phase, the optimal solution D is obtained after two iterations.

**Step 2** The quantity  $(\Delta f)_q$  is calculated for variables  $x_1$ ,  $x_2$  and  $x_5$ . It is found from the above table that the quantity  $(\Delta f)_q$  is maximum for both variables  $x_1$  and  $x_2$ . We choose the variable  $x_2$  (at random) to be the new basic variable. Next, we use the minimum ratio rule to choose the next new basic variable. The above table shows that the variable  $x_6$  must be replaced with the variable  $x_2$ .

**Step 3** Thus, we replace the variable  $x_6$  by  $x_2$  and form the next row-echelon tableau.

		0	0	0	0	0	-1	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
0	$x_3$	1	0	1	0	0	0	6
0	$x_4$	-1	0	0	1	2	-2	6
0	$x_2$	1	1	0	0	-1	1	2
$(\Delta f)_q$		0	0	0	0	0	-1	$f(x) = 0$

At this step, the solution is  $(0, 2)^T$  and is marked as B in Figure A.3.

**Step 2** Since the quantities  $(\Delta f)_q$  for all basic variables are non-positive, the optimal solution for the first phase is found. Therefore, we terminate the first phase and use this solution as the initial solution for the next phase. Here the solution corresponds to  $x_1 = 0$ ,  $x_2 = 2$ ,  $x_3 = 6$ ,  $x_4 = 6$ ,  $x_5 = 0$ , and

The artificial variable at this solution is zero. Thus, this solution is a feasible solution.

**Step 1** In the second phase, the objective function is the same as in the original problem:  $f(x) = 2x_1 + 3x_2$ . Since the artificial variable  $x_6$  was introduced to obtain an initial basic feasible solution, we need not continue with that variable in the second phase. Thus, we tabulate the new row-echelon form:

		2	3	0	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
0	$x_3$	1	0	1	0	0	6 $\frac{6}{0} = \infty$
0	$x_4$	-1	0	0	1	2	6 $\frac{6}{2} = 3 \leftarrow$
3	$x_2$	1	1	0	0	-1	2 $\frac{2}{0} = \infty$
$(\Delta f)_q$		-1	0	0	0	3	$f(x) = 6$

↑

At this step, the solution is  $(0, 2)^T$ .

**Step 2** The quantity  $(\Delta f)_q$  is calculated for all nonbasic variables in the above table. We observe that the variable  $x_5$  needs to be made basic. Using the minimum ratio rule, we also observe that the variable  $x_4$  needs to be exchanged with  $x_5$ . Notice that the objective function value is now calculated using the original function.

**Step 3** We create the new row-echelon form as follows:

		2	3	0	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
0	$x_3$	1.0	0	1	0.0	0	6 $\frac{6}{1} = 6 \leftarrow$
0	$x_5$	-0.5	0	0	0.5	1	3 $\frac{3}{0} = \infty$
3	$x_2$	0.5	1	0	0.5	0	5 $\frac{5}{0.5} = 10$
$(\Delta f)_q$		0.5	0	0	-1.5	0	$f(x) = 15$

↑

At this step, the solution is  $(0, 5)^T$  and is marked as C in Figure A.3. Notice that the objective function value at the current point is better than that in the previous iteration.

**Step 2** The quantity  $(\Delta f)_q$  is calculated for all nonbasic variables and only  $(\Delta f)_1$  is positive. Again, the minimum ratio rule suggests that the basic variable  $x_3$  needs to be made nonbasic.

**Step 3** Thus, we form the new row-echelon table.

		2	3	0	0	0
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
2	$x_1$	1	0	1.0	0.0	0
0	$x_5$	0	0	0.5	0.5	1
3	$x_2$	0	1	-0.5	0.5	0
$(\Delta f)_q$		0	0	-0.5	-1.5	0
						$f(x) = 18$

The solution obtained from the above table is  $(6, 2)^T$  with an objective function value equal to 18.

**Step 2** Since the quantities  $(\Delta f)_q$  for all nonbasic variables ( $x_3$  and  $x_4$ ) are non-positive, the optimal solution is obtained. Thus, we terminate the simplex method. Therefore, the final solution is  $(6, 2)^T$  as marked D in Figure A.3.

#### A.4 Big-M Method

The above method of using a dual phase method for handling ' $\geq$ '-type inequality constraints can be avoided by using the Big-M method in which a large negative value of the cost term is assigned to the artificial variables and the usual simplex method can be applied. Since the simplex method maximizes the objective function, the algorithm will attempt to make the artificial variable zero since a large negative cost value is assigned to this variable. This avoids the use of the dual phase LP method.

We illustrate the use of the Big-M method on the same problem used in the previous subsection. The objective function is written as follows:

$$f(\mathbf{x}) = 2x_1 + 3x_2 + 0x_3 + 0x_4 + 0x_5 - Mx_6.$$

Here,  $x_6$  is the artificial variable and  $M$  is a large positive number. We use  $M = 100$  for illustration purpose. The first tableau becomes as follows:

		2	3	0	0	0	-100	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
0	$x_3$	1	0	1	0	0	0	6
0	$x_4$	1	2	0	1	0	0	10
-1	$x_6$	1	1	0	0	-1	1	2
$(\Delta f)_q$		1	1	0	0	-1	0	$f(x) = -200$
↑								

The tableau is now updated and the following new tableau is obtained using the original simplex method:

		0	0	0	0	0	-1	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
0	$x_3$	1	0	1	0	0	0	6
0	$x_4$	-1	0	0	1	2	-2	6
3	$x_2$	1	1	0	0	-1	1	2
$(\Delta f)_q$		-1	0	0	0	3	-103	$f(x) = 6$

↑

The next tableau is given as follows:

		0	0	0	0	0	-1	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
0	$x_3$	1	0	1	0	0	0	6
0	$x_4$	-0.5	0	0	0.5	1	-1	3
3	$x_2$	0.5	1	0	0.5	0	0	5
$(\Delta f)_q$		0.5	0	0	-1.5	0	-100	$f(x) = 15$

↑

The next tableau finds the optimal solution:

		0	0	0	0	0	-1	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
0	$x_3$	1	0	1	0	0	0	6
0	$x_4$	0	0	0.5	0.5	1	-1	6
3	$x_2$	0	1	-0.5	0.5	0	0	2
$(\Delta f)_q$		0	0	-0.5	-1.5	0	-100	$f(x) = 18$

The optimal solution is  $(x_1, x_2) = (6, 2)^T$  with a function value of 18. This solution is the same as that found in the previous section.

## A.5 Algebraic Description of Simplex Method

In this section, we provide an algebraic description of the simplex method. Consider the following LP problem having  $J$  equality constraints:

$$\text{Maximize } f(x) = z = \mathbf{c}^T \mathbf{x},$$

subject to

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b}, \\ \mathbf{x} &\geq 0. \end{aligned} \tag{A.4}$$

At any iteration, there are  $J$  basic variables (set  $\mathbf{x}_B$ ) and  $(n - J)$  nonbasic variables (set  $\mathbf{x}_N$ ), such that  $\mathbf{x} = (\mathbf{x}_B, \mathbf{x}_N)^T$ . The matrix  $\mathbf{A}$  can also be partitioned as  $\mathbf{A} = (\mathbf{B}, \mathbf{N})$ , where  $\mathbf{B}$  contains columns of  $\mathbf{A}$  belonging to basic variables and  $\mathbf{N}$  contains columns of  $\mathbf{A}$  belonging to nonbasic variables. Similarly the cost vector  $\mathbf{c}$  can also be splitted as  $\mathbf{c} = (\mathbf{c}_B, \mathbf{c}_N)^T$ . The above LP problem can be written as follows:

$$\text{Maximize } f(x) = \mathbf{c}_B^T \mathbf{x}_B + \mathbf{c}_N^T \mathbf{x}_N,$$

subject to

$$\begin{aligned} \mathbf{B}\mathbf{x}_B + \mathbf{N}\mathbf{x}_N &= \mathbf{b}, \\ \mathbf{x}_B, \mathbf{x}_N &\geq 0. \end{aligned} \tag{A.5}$$

The simplex method identifies the basic variable set  $\mathbf{x}_B$  and manipulates the constraints in the following way:

$$\mathbf{x}_B + \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_N = \mathbf{B}^{-1}\mathbf{b}.$$

In each constraint (or, row in the simplex tableau), only one basic variable is present with a coefficient of one. From the above equation, the basic variable set can be written as follows:

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_N.$$

Now, let us consider the objective function  $z$  and rewrite in terms of nonbasic variables as follows:

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

If the term inside the bracket is positive, the nonbasic variables can be increased from zero to improve the objective function value  $z$ . Thus, such a solution cannot be optimal. So, a criterion for a solution to become optimum is that  $\bar{\mathbf{c}}^T = \mathbf{c}_N^T - \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{N}$  is non-positive (or,  $\bar{\mathbf{c}}^T \leq 0$ ) and at the optimal solution the objective function value is  $\mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b}$ .

Let us now discuss the inequality constraint case ( $\mathbf{Ax} \leq \mathbf{b}$ ). In such a case, first, the slack variables are added to find a set of basic variable set. Let us say that the slack variables are represented as  $\mathbf{x}_S$  and there are  $J$  such variables, one for each constraint. At the initial starting iteration, the following simplex tableau is formed:

		$\mathbf{c}_B^T$	$\mathbf{c}_N^T$	$\mathbf{0}^T$		
		$\mathbf{x}_B^T$	$\mathbf{x}_N^T$	$\mathbf{x}_S^T$		
$\mathbf{0}$	$\mathbf{x}_S$	$\mathbf{B}$	$\mathbf{N}$	$\mathbf{I}$	$\mathbf{b}$	
	$(\Delta f)_q$	$\mathbf{c}_B^T$	$\mathbf{c}_N^T$	$\mathbf{0}^T$	$f(x) = 0$	

The basic variable set is  $\mathbf{x}_S$  and the combined  $\mathbf{x} = (\mathbf{x}_N, \mathbf{x}_B)^T$  is the nonbasic variable set. The parameter  $(\Delta f)_q$  are cost terms and are usually positive.

As the simplex algorithm proceeds and get into the final tableau representing the optimal solution, the above scenario changes. In most cases,  $J$  variables from  $\mathbf{x}$  (and not from  $\mathbf{x}_S$  usually) become basic variables. Let us call this basic variable set as  $\mathbf{x}_B$  and the remaining variables ( $\mathbf{x}_N$ ) together with the slack variable set  $\mathbf{x}_S$  become the combined nonbasic variable set. Let us write the final tableau in algebraic form, as shown in Table A.1. There are several important observations that can be made from this final tableau.

**Table A.1** Algebraic Terms of the Final Tableau in a Simplex Algorithm.

		$\mathbf{c}_B^T$	$\mathbf{c}_N^T$	$\mathbf{0}^T$	
	Basic	$\mathbf{x}_B^T$	$\mathbf{x}_N^T$	$\mathbf{x}_S^T$	
$\mathbf{c}_B$	$\mathbf{x}_B$	I	$\mathbf{B}^{-1}\mathbf{N}$	$\mathbf{B}^{-1}$	$\mathbf{B}^{-1}\mathbf{b}$
$(\Delta f)_q$	$\mathbf{0}^T$	$\mathbf{c}_N^T - \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{N}$	$-\mathbf{c}_B^T \mathbf{B}^{-1}$	$f(x) = \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b}$	

- (i) A solution is feasible if  $\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} \geq 0$ .
- (ii) A solution is optimal (maximum), if the reduced cost values  $\bar{\mathbf{c}}_N$  is non-positive, or,  $\mathbf{c}_N^T - \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{N} \leq 0$ . This is true because at the final tableau all quantities for  $(\Delta f)_q$  is non-positive.
- (iii) The objective function value is  $\mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b}$ .
- (iv) The shadow price vector (or, the Lagrange multiplier vector) for constraints is given by  $\mathbf{c}_B^T \mathbf{B}^{-1}$  and can be directly obtained from the entries in the  $(\Delta f)_q$  row for slack variable set  $\mathbf{x}_S$  at the final tableau.

The above information will be used in the discussions on sensitivity analysis in Section A.7.

Let us identify the above-mentioned terms for the final tableau (see page 376) for the example problem given in Equation (A.3):

$$\mathbf{B}^{-1} = \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix},$$

$$\mathbf{c}_B^T \mathbf{B}^{-1} = \begin{bmatrix} 0.5 & 1.5 \end{bmatrix},$$

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} = \begin{bmatrix} 6 \\ 2 \end{bmatrix},$$

$$f(\mathbf{x}^*) = \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b} = 18.$$

They indicate that the optimal solution is  $\mathbf{x}^* = \mathbf{x}_B$  with a function value of 18. Also, the shadow price for two constraints are 0.5 and 1.5, respectively, as obtained from the term  $\mathbf{c}_B^T \mathbf{B}^{-1}$ . We shall use  $\mathbf{B}^{-1}$  and  $\mathbf{c}_B^T \mathbf{B}^{-1}$  in a later section. Further details can be obtained from Lundgren et al. (2011).

## A.6 Revised Simplex Method

The simplex method described earlier is computationally expensive, as it requires expensive row-echelon update of the entire  $J \times (N - J)$  matrix. The algebraic representation of the simplex method can be utilized to re-write a computationally faster algorithm. This is what is achieved in the revised simplex method.

### Algorithm

**Step 1** Choose a basic feasible solution  $\mathbf{x}_B^{(0)}$  and a basic matrix  $\mathbf{B}^{(0)}$ . The remaining variable set is  $\mathbf{x}_N^{(0)}$  and corresponding matrix  $\mathbf{N}^{(0)}$ . Set an iteration counter  $t = 0$ . All quantities with a superscript of  $(t)$  get updated with the iteration counter.

**Step 2** Compute the reduced cost,  $(\bar{c}_N^{(t)})^T = c_N^{(t)T} - \mathbf{c}_B^{(t)T} (\mathbf{B}^{(t)})^{-1} \mathbf{N}^{(t)}$ .

The matrix  $\mathbf{N}^{(t)}$  is constructed by collecting columns of nonbasic variables from the original  $[\mathbf{A}|I]$  matrix. The vectors  $c_N^{(t)T}$  and  $\mathbf{c}_B^{(t)}$  are chosen from the original cost vector  $\mathbf{c}$ .

**Step 3** If  $\bar{c}_N^{(t)} \leq \mathbf{0}$ , the optimal solution  $(\mathbf{x}^* = (\mathbf{x}_B^{(t)}, \mathbf{x}_N^{(t)})^T)$  is obtained. Terminate the algorithm. Else proceed to Step 4.

**Step 4** Identify the entering basic variable using

$$p = \operatorname{argmax}_j \{\bar{c}_j^{(t)} \mid \bar{c}_j^{(t)} > 0, \text{ and for all nonbasic variables } x_j\}.$$

The variable  $x_p$  becomes the new basic variable.

**Step 5** Update  $p$ -th column using  $\mathbf{a}_p^{(t)} = (\mathbf{B}^{(t)})^{-1} \mathbf{a}_p$ , where  $\mathbf{a}_p$  is the  $p$ -th column in the original  $[\mathbf{A}|I]$  matrix. Next, compute

$$r = \operatorname{argmin}_i \left\{ \frac{b_i^{(t)}}{a_{ip}^{(t)}} \mid a_{ip}^{(t)} > 0 \right\}.$$

Variable  $x_r$  is replaced by  $x_p$  in the basic variable set.

**Step 6** Update  $\mathbf{x}_B^{(t+1)}$  by replacing  $x_r$  with  $x_p$ . Update  $\mathbf{x}_N^{(t+1)}$ ,  $\mathbf{c}_B^{(t+1)}$  and  $\mathbf{c}_N^{(t+1)}$ . Replace column  $r$  with column  $p$  to form the new basis matrix  $\mathbf{B}^{(t+1)}$ . Update  $\mathbf{N}^{(t+1)}$ . Then calculate  $\mathbf{x}_B^{(t+1)} = (\mathbf{B}^{(t+1)})^{-1} \mathbf{b}$ . The objective value

is  $f(\mathbf{x}^{(t+1)}) = \mathbf{c}_B^{(t+1)T} \mathbf{x}_B^{(t+1)}$ . Increment iteration counter by one ( $t = t + 1$ ) and go to Step 2.

The above revised simplex algorithm is exactly the same in principle to the original simplex algorithm, except that it is computationally faster. Since all the above operations involve a major computation in computing inverse of  $\mathbf{B}$ -matrix of size  $J \times J$ , the overall computation is much smaller than that needed in the original simplex algorithm. While solving large-sized problems (such as problems involving hundreds of thousands of variables and thousands of constraints), the computational advantage will be apparent.

We illustrate the working of the above procedure on the example problem given in Equation (A.3).

**Step 1** There are two basic variables  $\mathbf{x}_B^{(0)} = (x_3, x_4)^T$  and two nonbasic variables  $\mathbf{x}_N^{(0)} = (x_1, x_2)^T$  at iteration zero ( $t = 0$ ). The matrices are as follows:

$$\mathbf{B}^{(0)} = (\mathbf{a}_3, \mathbf{a}_4) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{N}^{(0)} = (\mathbf{a}_1, \mathbf{a}_2) = \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix}.$$

**Step 2** Reduced cost is computed as follows:

$$\begin{aligned} (\bar{c}_N^{(0)})^T &= c_N^{(0)T} - \mathbf{c}_B^{(0)T} \left( \mathbf{B}^{(0)} \right)^{-1} \mathbf{N}^{(0)} = (2, 3)^T - (0, 0) \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} \\ &= (2, 3). \end{aligned}$$

In other words,  $(\bar{c}_1, \bar{c}_2)^{(0)} = (2, 3)$ .

**Step 3** Since both  $\bar{c}_j$  values are non-negative, we do not terminate the algorithm.

**Step 4** We compute  $p = \text{argmax}\{\bar{c}_1^{(0)}, \bar{c}_2^{(0)}\} = 2$ . Thus, variable  $x_2$  should become a basic variable.

**Step 5** First, we compute

$$\mathbf{a}_2^{(0)} = \left( \mathbf{B}^{(0)} \right)^{-1} \mathbf{a}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}.$$

Next, we identify  $r = \text{argmin} \left\{ \frac{6}{0} \text{ for } x_3, \frac{10}{2} \text{ for } x_4 \right\} = 4$ . Thus, variable  $x_4$  will get replaced by variable  $x_2$ .

**Step 6** We update the variable vectors as follows:  $\mathbf{x}_B^{(1)} = (x_3, x_2)^T$  and  $\mathbf{x}_N^{(1)} = (x_1, x_4)^T$ . Also,  $\mathbf{c}_B^{(1)} = (0, 3)^T$  and  $\mathbf{c}_N^{(1)} = (2, 0)^T$ . Now,

$$\mathbf{B}^{(1)} = (\mathbf{a}_3, \mathbf{a}_2) = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}.$$

Its inverse is

$$(\mathbf{B}^{(1)})^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}.$$

The basic variable values are computed as follows:

$$\mathbf{x}_B^{(1)} = (\mathbf{B}^{(1)})^{-1} \mathbf{b} = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 6 \\ 10 \end{bmatrix} = \begin{bmatrix} 6 \\ 5 \end{bmatrix}.$$

The function value is

$$f(\mathbf{x}^{(1)}) = \mathbf{c}_B^{(1)T} \mathbf{x}_B^{(1)} = (0, 3) \begin{bmatrix} 6 \\ 5 \end{bmatrix} = 15.$$

One iteration of the revised simplex method is over and we now move to Step 2.

**Step 2** We compute the reduced cost

$$(\bar{c}_N^{(1)})^T = (2, 0) - (0, 3) \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = (0.5, -1.5).$$

**Step 3** Since  $\bar{c}_1 > 0$ , we do not terminate the algorithm.

**Step 4**  $p = \text{argmax}\{0.5 \text{ for } x_1, -1.5 \text{ for } x_4\} = 1$ . Thus, variable  $x_1$  should now become the basic variable.

**Step 5** The new column 1 is as follows:

$$\mathbf{a}^{(1)} = (\mathbf{B}^{(1)})^{-1} \mathbf{a}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}.$$

The index  $r = \text{argmin} \left\{ \frac{6}{1} \text{ for } x_3, \frac{5}{0.5} \text{ for } x_2 \right\} = 3$ . Thus, variable  $x_3$  will get replaced by  $x_1$ .

**Step 6** The new basic variable set is  $\mathbf{x}_B^{(2)} = (x_1, x_2)^T$  and  $\mathbf{x}_N^{(2)} = (x_3, x_4)^T$ . Also,  $\mathbf{c}_B^{(2)} = (2, 3)^T$  and  $\mathbf{c}_N^{(2)} = (0, 0)^T$ . Now,

$$\mathbf{B}^{(2)} = (\mathbf{a}_1, \mathbf{a}_2) = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}.$$

Its inverse is

$$(\mathbf{B}^{(1)})^{-1} = \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix}.$$

The basic variable values are computed as follows:

$$\mathbf{x}_B^{(2)} = (\mathbf{B}^{(2)})^{-1} \mathbf{b} = \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 6 \\ 10 \end{bmatrix} = \begin{bmatrix} 6 \\ 2 \end{bmatrix}.$$

The function value is

$$f(\mathbf{x}^{(2)}) = \mathbf{c}_B^{(2)T} \mathbf{x}_B^{(2)} = (2, 3) \begin{bmatrix} 6 \\ 2 \end{bmatrix} = 18.$$

The second iteration of the revised simplex method is also over and we now move to Step 2 for the third iteration ( $t = 2$ ).

**Step 2** The reduced cost is

$$(\bar{c}_N^{(2)})^T = (0, 0) - (2, 3) \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = (-0.5, -1.5).$$

**Step 3** Since both elements of  $\bar{c}_N^{(2)} < 0$ , we have found the optimal solution and we now terminate the algorithm and declare  $\mathbf{x}^* = (x_1, x_2, x_3, x_4)^{(2)} = (6, 2, 0, 0)^T$  as the final solution with a function value of 18.

The revised simplex method is easier to convert to a computer algorithm and as mentioned before, it makes a computationally quicker implementation of the original simplex algorithm.

## A.7 Sensitivity Analysis for Linear Programming

Sensitivity analysis refers to a post-optimality analysis that is usually performed to gather better understanding of the nature of the obtained optimum. At the optimum point, the following sensitivity analysis can be performed:

- (i) *The sensitivity of the optimal solution with a change in the right-hand side (RHS) parameters ( $b_k$  values):* The right-hand side parameter usually signifies the capacity or resource values. After obtaining the optimal solution, the user may be interested in finding how the solution will change if the capacity or resource values are increased or decreased. Thus, such a sensitivity analysis has tremendous practical significance. This analysis is similar in principle to the sensitivity analysis procedure discussed in the case of nonlinear programming problem in Section 4.4.
- (ii) *The sensitivity of the optimal solution with respect to a change in the cost coefficient ( $c_B$ ) in the objective function:* If the prices are increased or decreased, the user may be interested in knowing whether the optimal solution changes and if changes, by how much.

One other important matter of interest in both the above cases is to know the extent of changes (either in RHS or cost terms) from their current values which will not alter the current optimal solution. In some studies, a change in both RHS and cost terms may be simultaneously changed and their effects on the optimal solutions are required to be known. We discuss each of the two cases in the following subsections.

#### A.7.1 Sensitivity with Respect to RHS of Constraints

Here, we ask the question: ‘When the right-hand side (RHS) of constraints is changed, within what range the basic variable set  $\mathbf{x}_B$  remains feasible?’ We answer this question from the observations made at the end of Section A.5. As the RHS of  $j$ -th constraint is changed by  $\Delta b_j$  such that the new RHS vector is  $(\mathbf{b} + \Delta\mathbf{b})$ , the new optimal solution becomes  $\mathbf{x}_B = \mathbf{B}^{-1}(\mathbf{b} + \Delta\mathbf{b})$ . For a solution to remain feasible,  $\mathbf{x}_B \geq 0$ , meaning that

$$\begin{aligned}\mathbf{x}_B &= \mathbf{B}^{-1}(\mathbf{b} + \Delta\mathbf{b}) \geq 0, \\ &= \mathbf{B}^{-1}\mathbf{b} + \mathbf{B}^{-1}\Delta\mathbf{b} \geq 0, \\ &= \mathbf{x}_B^{\text{old}} + \mathbf{B}^{-1}\Delta\mathbf{b} \geq 0.\end{aligned}\tag{A.6}$$

Equation (A.6) can be used to find the allowable range in the change in RHS parameters so that  $\mathbf{x}_B$  remains feasible, provided we have an easier way to compute the matrix  $\mathbf{B}^{-1}$ . The algebraic terms in Table A.1 helps us to find this matrix. Notice that this matrix is already computed at the final tableau from the columns of the slack variable set  $\mathbf{x}_S$  corresponding to constraint rows. Note that the new optimum solution becomes  $\mathbf{x}_B$  given in Equation (A.6) and the corresponding objective value is  $\mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b}$ . We take an example to illustrate the above procedure.

Let us consider the example problem given in Equation (A.3). Note that for this problem, the quantities  $\mathbf{x}_B$  and  $\mathbf{B}^{-1}$  are as follows:

$$\mathbf{x}_B = \begin{bmatrix} 6 \\ 2 \end{bmatrix}, \quad \mathbf{B}^{-1} = \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix}.$$

Let us also assume that we perturb constraints 1 and 2 by  $\Delta b_1$  and  $\Delta b_2$ , respectively. Equation (A.6) implies that in order for the corresponding optimal solution to remain feasible, the following conditions must be met:

$$\begin{aligned} \begin{bmatrix} 6 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} \Delta b_1 \\ \Delta b_2 \end{bmatrix} &\geq 0, \\ \text{or, } \begin{bmatrix} 6 + \Delta b_1 \\ 2 - \frac{1}{2}\Delta b_1 + \frac{1}{2}\Delta b_2 \end{bmatrix} &\geq 0. \end{aligned}$$

The above conditions yields the following:

$$\Delta b_1 \geq -6, \quad (\text{A.7})$$

$$\Delta b_2 \geq -4 + \Delta b_1. \quad (\text{A.8})$$

The above two conditions remain as important restrictions for the optimal solution to remain feasible due to changes in the RHS of both constraints. For a fixed change in  $b_2$  (say by  $\Delta b_2$ ), there is a range of  $\Delta b_1$  that will ensure feasibility of the resulting optimal solution:

$$-6 \leq \Delta b_1 \leq 4 + \Delta b_2.$$

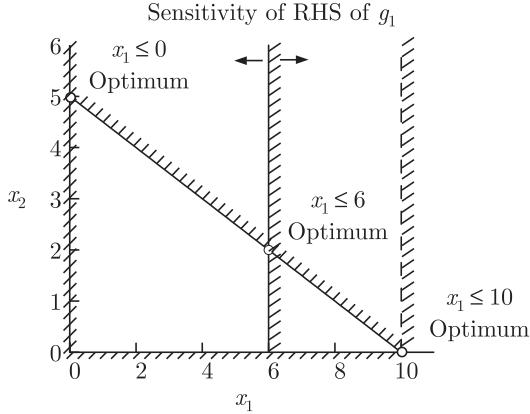
For a change in the RHS of constraint 1 alone,  $\Delta b_2 = 0$  and the allowable change in  $b_1$  is  $-6 \leq \Delta b_1 \leq 4$  or,  $0 \leq b_1 \leq 10$ . For this case, the optimal solution changes (for a fixed  $b_2 = 10$ ) as follows:

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} \in \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix} \left[ \begin{bmatrix} 0 \\ 10 \end{bmatrix}, \begin{bmatrix} 10 \\ 10 \end{bmatrix} \right] = \left[ \begin{bmatrix} 0 \\ 5 \end{bmatrix}, \begin{bmatrix} 10 \\ 0 \end{bmatrix} \right].$$

The function values change as follows:

$$f(\mathbf{x}^*) = \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b} \in \begin{bmatrix} 2 & 3 \end{bmatrix} \left[ \begin{bmatrix} 0 \\ 5 \end{bmatrix}, \begin{bmatrix} 10 \\ 0 \end{bmatrix} \right] = \begin{bmatrix} 15, & 20 \end{bmatrix}.$$

The above sensitivity analysis can also be verified from Figure A.4. Keeping the second constraint identical, if the first constraint is allowed to change from its original value  $b_1 = 6$ , the allowable range of values of  $b_1$  is  $[0, 10]$ . When  $b_1 = 0$ , the constraint becomes  $x_1 \leq 0$ , meaning the solutions along the  $x_2$ -axis and in  $0 \leq x_2 \leq 5$  are feasible. The optimal solution is the point  $(0, 5)^T$  with a function value of  $2 \times 0 + 3 \times 5$  or 15. The estimated function value at this point is also found to be 15 as shown above. The RHS parameter  $b_1$  cannot be reduced below zero. On the other hand, when  $b_1$  is increased to 10,



**Figure A.4** Change of right-hand side coefficient of the first constraint and its effect on the optimum.

constraint 1 is redundant and the optimal solution is  $(10, 0)$  with a function value of  $2 \times 10 + 3 \times 0$  or 20, which matches with the estimated value.

However, if  $b_1$  is increased beyond 10 (say  $b_1 = 12$ ), then also constraint 1 is redundant and the optimal solution is still  $(10, 0)$  with  $f(\mathbf{x}) = 20$ . But the prediction of the objective value using  $\mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b}$  comes out to be 21. Thus, if the change in the RHS parameters are kept within their allowable range, as given in Equation (A.6), the predicted solution and objective value will be true and another application of the simplex procedure for the new values of the RHS vector can be avoided.

### A.7.2 100% Rule for Simultaneous Changes in RHS Parameters

When the RHS parameters for more than one constraint is changed simultaneously, the conditions for checking if the perturbed solution is feasible are different from that described above. Let us say that for the  $j$ -th constraint,  $\Delta b_j$  is the desired perturbation,  $U_j$  is the maximum feasible increase of  $b_i$  for keeping the solution feasible, and  $L_j$  is the minimum feasible decrease of  $b_i$  for keeping the solution feasible. Let us also define a ratio  $r_j$  for the perturbation in the  $j$ -th constraint as follows:

$$r_j = \begin{cases} \frac{\Delta b_j}{U_j}, & \text{if } \Delta b_j \geq 0, \\ \frac{-\Delta b_j}{L_j}, & \text{if } \Delta b_j < 0. \end{cases} \quad (\text{A.9})$$

For multiple changes in the RHS coefficient,  $r_i$  is computed for each such constraint and if the  $\sum_j r_j$  is less than or equal to one, the resulting solution will be feasible. However, if  $\sum_j r_j > 1$ , the solution may or may not remain feasible.

In the above example, for a sole change in  $b_1$ , we obtained the allowable bound for  $\Delta b_1$  as  $[-6, 4]$ . Thus,  $L_1 = -6$  and  $U_1 = 4$ . Similarly, if  $b_2$  is changed alone,  $\Delta b_2 \in [-4, \infty]$  is allowed. Thus,  $L_2 = -4$  and  $U_2 = \infty$ . Now, if  $\Delta b_1 = -2$  and  $\Delta b_2 = -2$  are chosen, the constraints become as follows:

$$\begin{aligned}x_1 &\leq 4, \\x_1 + 2x_2 &\leq 8.\end{aligned}$$

For both changes mentioned above,  $r_1 = (-2)/(-6) = 1/3$  and  $r_2 = (-2)/(-4) = 1/2$ . The sum  $r_1 + r_2 = 5/6$  which is less than one. Hence, the original optimal solution will remain feasible under these changes in the RHS values. In fact, for the above two constraints, the optimal solution is  $(4, 2)^T$  with a function value equal to 14. It is interesting to note that for a change in only one constraint, the above 100% rule ensures that a change within its allowable range will always produce a feasible solution.

### A.7.3 Sensitivity with Respect to Cost Terms

After the optimal solution is obtained, one may be interested in knowing how would the solution and its function value change if the cost terms ( $\mathbf{c}_B$  and  $\mathbf{c}_N$ ) are changed. It could be that unlike in the case of a change in the RHS parameters, here, the optimal solution does not change for a perturbation in the cost terms. Then, one may be interested in knowing what are the ranges of change in cost terms that will not alter the original optimal solution.

For this purpose, we recall that the condition for optimality is when  $\bar{\mathbf{c}}_N \leq 0$ . Let us say that all cost terms are changed, such that the new  $\bar{\mathbf{c}}_N$  is calculated as follows:

$$\bar{\mathbf{c}}_N = (\mathbf{c}_N^T + \Delta \mathbf{c}_N^T) - (\mathbf{c}_B^T + \Delta \mathbf{c}_B^T) \mathbf{B}^{-1} N \leq 0. \quad (\text{A.10})$$

Since the solution  $\mathbf{x}_B$  does not change, satisfaction of Equation A.10 will provide the range of cost terms that will not affect the optimal solution. The revised objective function becomes  $(\mathbf{c}_B + \Delta \mathbf{c}_B)^T \mathbf{x}_B$ .

Let us take the example problem given in Equation (A.3). Say, we change  $c_1$  and  $c_2$  terms by  $\Delta c_1$  and  $\Delta c_2$ , respectively. Recall that in this problem, there is no  $\mathbf{c}_N$  term. Therefore, the reduced cost term becomes as follows:

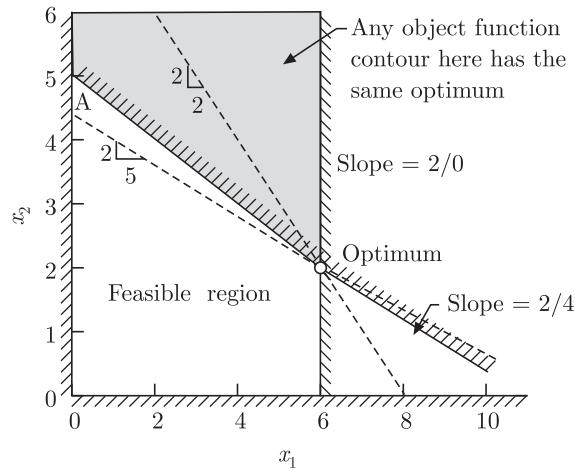
$$\begin{aligned}\bar{\mathbf{c}}_N &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 + \Delta c_1 & 3 + \Delta c_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix} \\&= \begin{bmatrix} -0.5 - \Delta c_1 + 0.5 \Delta c_2 \\ -1.5 - 0.5 \Delta c_2 \end{bmatrix} \leq 0.\end{aligned}$$

The above yields the following conditions:

$$\Delta c_1 \geq 0.5\Delta c_2 - 0.5,$$

$$\Delta c_2 \geq -3.$$

For a fixed  $\Delta c_1$ , there is a range of values of  $\Delta c_2$  which will result in an identical optimal solution:  $-3 \leq \Delta c_2 \leq 2\Delta c_1 + 1$ . For example, if the cost term for the first variable is unchanged (that is,  $\Delta c_1 = 0$ ), the range of allowable values for  $\Delta c_2$  is  $-3 \leq \Delta c_2 \leq 1$  or,  $0 \leq c_2 \leq 4$ . When  $c_2 = 0$ , there is no importance of the second variable, the optimal solution can still be  $(6, 2)^T$  (in fact, any point for  $x_1 = 6$  and  $0 \leq x_2 \leq 2$  is an optimal solution). When  $c_2 = 4$ , all solutions on the constraint 2 within  $x_1 \in [0, 6]$  are optimal, including the original solution  $(6, 2)^T$ . If a change  $\Delta c_2 = -2$  is chosen, the optimal solution is still  $(6, 2)^T$ , but the function value changes by  $\mathbf{c}_B^T(0, -2)^T$  or  $-1.5$ . Figure A.5 depicts these scenarios. Any slope of the contour line of the objective function in the range  $(2/0, 2/4)$  keeps the optimum at  $(6, 2)^T$ .



**Figure A.5** Change of cost term for  $x_2$  and its effect on the optimum. Cost term  $c_2$  in the range  $[0, 4]$  does not change the position of the optimum for  $c_1 = 2$ .

A change in  $c_2$  outside the range  $(0, 4)$  (and no change in  $c_1$ ) changes the original optimal solution. For example, if  $c_2 = 5$  is chosen, solution  $(0, 5)^T$  becomes the new optimal solution. The contour of the objective function has a slope  $2/5$  and as shown in the figure, the original optimum  $(6, 2)^T$  does not remain optimum any more, instead the point  $(0, 5)^T$  becomes the new optimum.

#### A.7.4 100% Rule for Simultaneous Changes in Cost Terms

There also exists a 100% rule for changes in multiple cost terms simultaneously. Let us say that there is a change  $\Delta c_i$  anticipated in the

$i$ -th cost term  $c_i$ . Let us also say that  $U_i$  is the maximum allowable increase in  $\Delta c_i$  to keep optimality, and  $L_i$  is the minimum allowable decrease in  $\Delta c_i$  to keep optimality. Then the ratio  $r_i$  is computed as follows:

$$r_i = \begin{cases} \frac{\Delta c_i}{U_i}, & \text{if } \Delta c_i \geq 0, \\ \frac{-\Delta c_i}{L_i}, & \text{if } \Delta c_i < 0. \end{cases} \quad (\text{A.11})$$

If the combined ratio ( $\sum_i r_i$ ) is smaller than or equal to one, the changes in cost terms do not change the original optimal solution, otherwise nothing can be said about the optimality of the original solution in the new context.

For the example problem considered in the previous subsection, condition A.10 yields:  $\Delta c_1 \geq -0.5$  and  $-3 \leq \Delta c_2 \leq 1$ . Therefore,  $L_1 = -0.5$ ,  $U_1 = \infty$ ,  $L_2 = -3$  and  $U_2 = 1$ . For desired changes of  $\Delta c_1 = -0.5$  and  $\Delta c_2 = 0.5$ , the respective  $r_1 = (-0.5)/(-0.5) = 1$  and  $r_2 = 0.5/1 = 0.5$ . The combined change is greater than one, hence nothing can be said about the optimality of the solution. The above changes make the objective function  $f(\mathbf{x}) = 1.5x_1 + 3.5x_2$ . The original solution  $(6, 2)^T$  with a function value of 16 is no more optimal, as the solution  $(0, 5)^T$  has a better function value of 16.5. However, if a  $\Delta c_1 = -0.2$  is used, making  $r_1 = (-0.2)/(-0.5) = 0.4$  and  $r_1 + r_2 = 0.9 < 1$ . The solution  $(6, 2)^T$  has a function value of 17.8, while the solution  $(0, 5)^T$  has a function value of 16.5. Thus, the 100% rule correctly predicts the optimality of the original solution.

## A.8 Duality Theory in Linear Programming

In Section 4.2, we have mentioned that for every primal problem, there exists a dual problem in which there are as many variables as there are constraints in the original problem. In some occasions, solving a dual problem is easier. Such an indirect solution to a problem is beneficial if there is a way one can establish a relationship between the primal and dual problems. Since LP problems are linear, a relationship between primal and dual problems exists and we can exploit that relationship to solve the LP problems indirectly.

Here, we formulate the dual problem of a primal LP problem in a different way. Consider the LP (primal) problem given in Equation (A.3). Let us say that we multiply the second constraint by two and we obtain the following:

$$2x_1 + 4x_2 \leq 20.$$

Note that we are required to maximize the objective function  $f(\mathbf{x}) = 2x_1 + 3x_2$ . Since both  $x_1$  and  $x_2$  are non-negative, we can safely say that the optimal objective function value cannot be greater than 20. This is because, the sum of  $2x_1$  and  $4x_2$  cannot be more than 20, hence the sum of  $2x_1$  and  $3x_2$  cannot also be greater than 20. Although from the first constraint alone we may never get such a bound, we may try the following. Since the upper bounds of some linear combinations of variables are known from the constraints, we

can use two variables (called dual variables), say  $y_1$  and  $y_2$ , to combine the constraints as follows:

$$\begin{aligned} y_1(x_1) + y_2(x_1 + 2x_2) &\leq 6y_1 + 10y_2, \\ (y_1 + y_2)x_1 + 2y_2x_2 &\leq 6y_1 + 10y_2. \end{aligned}$$

Our goal is to choose  $y_1$  and  $y_2$  such that the left side expression approaches the objective function  $f(\mathbf{x}) = 2x_1 + 3x_2$  but by keeping the coefficients not smaller than 2 and 3, respectively. That is, we would like to satisfy the following constraints:

$$y_1 + y_2 \geq 2,$$

$$2y_2 \geq 3.$$

For a set of values of  $y_1$  and  $y_2$ , the right-hand side ( $6y_1 + 10y_2$ ) puts the upper bound of  $f(\mathbf{x})$ . Since our goal is to come close to the optimum of  $f(\mathbf{x})$ , the following optimization problem should give us the requisite values of  $y_1$  and  $y_2$ :

$$\text{Minimize } f(x) = 6y_1 + 10y_2,$$

subject to

$$\begin{aligned} y_1 + y_2 &\geq 2, \\ 2y_2 &\geq 3, \\ y_1, y_2 &\geq 0. \end{aligned} \tag{A.12}$$

This problem is known as the dual problem to the primal problem given in Equation (A.3).

Thus, for the following primal LP problem (maximization):

$$\text{Maximize } f(\mathbf{x}) = \mathbf{c}^T \mathbf{x},$$

subject to

$$\begin{aligned} \mathbf{A}\mathbf{x} &\leq \mathbf{b}, \\ \mathbf{x} &\geq 0, \end{aligned} \tag{A.13}$$

we can write the corresponding dual LP problem (minimization), as follows:

$$\text{Minimize } w(\mathbf{y}) = \mathbf{b}^T \mathbf{y},$$

subject to

$$\begin{aligned} \mathbf{A}^T \mathbf{y} &\geq \mathbf{c}, \\ \mathbf{y} &\geq 0. \end{aligned} \tag{A.14}$$

The primal problem has  $n$  variables and  $J$  constraints, whereas the dual problem has  $J$  variables and  $n$  constraints. Notice how the cost term  $\mathbf{c}$  and RHS coefficient vector  $\mathbf{b}$  are interchanged between the two problems. Also, the primal problem is a maximization type and the dual problem is a minimization type. One other vital difference is that while constraints in the primal problem are ' $\leq$ '-type, that of the dual problem are of ' $\geq$ '-type.

Besides the differences, there are certain relationships between the two problems which are of great importance in solving LP problems in general.

The *weak duality* theorem states that for a feasible solution  $\hat{\mathbf{x}}$  to the primal problem and for a feasible solution  $\hat{\mathbf{y}}$  to the dual problem,

$$\mathbf{c}^T \hat{\mathbf{x}} \leq \mathbf{b}^T \hat{\mathbf{y}}. \quad (\text{A.15})$$

This means that a feasible primal solution has always smaller or equal objective function value than the objective value of any feasible dual solution. If somehow a feasible dual solution can be identified for a primal problem, its dual objective value can be taken as the upper bound of the primal optimal function value. There exists a strong duality concept that is more useful.

The *strong duality* theorem states that if one of the problems has a bounded optimal solution, then the same holds for the other problem as well, and importantly the two optimal objective function values are equal, that is,

$$\mathbf{c}^T \mathbf{x}^* = \mathbf{b}^T \mathbf{y}^*. \quad (\text{A.16})$$

This theorem suggests that if we solve the dual problem, the optimal objective value of the primal problem is identical to the optimal objective value of the dual problem.

We now develop another interesting relationship between the two problems at their respective optimum ( $\mathbf{x}^*$  and  $\mathbf{y}^*$ ). Recall from the algebraic analysis of the simplex method that

$$f(\mathbf{x}^*) = \mathbf{c}_B^T \mathbf{x}_B^* = \mathbf{c}_B^T \mathbf{B}^{-1} \mathbf{b}.$$

Also at the dual optimum, we can write

$$w(\mathbf{y}^*) = \mathbf{b}^T \mathbf{y}_B^* = \mathbf{y}_B^{*T} \mathbf{b}.$$

Comparing the above two equations using Equation (A.16), we obtain

$$\mathbf{y}_B^{*T} = \mathbf{c}_B^T \mathbf{B}^{-1}. \quad (\text{A.17})$$

This means that the optimal dual solution can be obtained from the final simplex tableau (Table A.1) of the primal LP procedure. Recall that  $-\mathbf{c}_B^T \mathbf{B}^{-1}$  vector appears under the column of slack variables  $\mathbf{x}_S$  for the  $(\Delta f)_q$  row. We now demonstrate the interchangeability of both problems through the same example problem given in Equation (A.3).

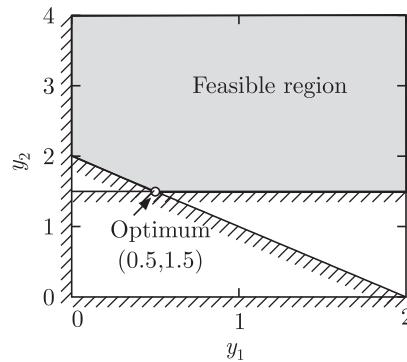
The final tableau using the simplex method is reproduced here for convenience (Table A.2): Here, the slack variables are  $(x_3, x_4)^T$ . Thus, the

**Table A.2** Final Tableau of LP Problem Given in Equation (A.3)

		2	3	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
2	$x_1$	1	0	1.0	0.0	6
3	$x_2$	0	1	-0.5	0.5	2
$(\Delta f)_q$				0	0	-0.5 -1.5
				$f(x) = 18$		

vector under  $\mathbf{x}_S$  in the  $(\Delta f)_q$  row is  $(-0.5, -1.5)$ . In Table A.1, we have marked this vector as  $-\mathbf{c}_B^T \mathbf{B}^{-1}$ . Thus, the negative of this vector is nothing but the optimal dual solution  $\mathbf{y}_B^* = (\mathbf{c}_B^T \mathbf{B}^{-1})^T = (0.5, 1, 5)^T$ . Thus, without solving the dual problem, we can predict the solution of the dual problem using the final tableau of the primal LP procedure.

We now solve the dual problem graphically and investigate if the above-obtained solution is indeed the dual solution. The dual problem is formulated in Equation (A.12). Figure (A.6) shows the feasible space and the corresponding optimal (dual) solution. Since the function  $6y_1 + 10y_2$

**Figure A.6** Graphical representation of the dual problem and its solution.

is to be minimized, the dual solution is  $(0.5, 1.5)^T$  with a function value of  $6 \times 0.5 + 10 \times 1.5$  or 18, which is identical to the primal optimal objective function value.

It is interesting to note that the dual problem given in Equation (A.12) is also an LP and one can attempt to solve the dual problem using the simplex procedure described before. However, there are a number of difficulties with the dual problem being solved using the simplex method:

- (i) First, the objective function is of minimization type. This can be handled by multiplying the objective function by  $-1$  and maximizing the function as usual, but then the cost terms of the maximization function are all negative.

- (ii) Second, the constraints are of ‘ $\geq$ ’-type, thereby requiring to use the artificial variables and resorting to the dual phase method.

Although the Big-M method discussed in Section A.4 can be employed, in the following section, we suggest another method that makes such problem solving relatively easier and also allows an easier way to perform a sensitivity analysis.

### A.9 Dual-Simplex Method

We may now call the previous simplex method (described in Section A.2) as the primal simplex method, since it was suggested to be used for solving primal LP problems. The simplex method we describe in this section is specially designed for solving problems that are infeasible at the current point, such as at the initial infeasible solution (having  $\mathbf{x}_B \leq 0$ ). If the dual of such a problem is feasible, the primal simplex algorithm described earlier can be changed to solve the same problem. We illustrate the so-called dual-simplex method in this section. The dual-simplex method is also helpful in performing sensitivity analysis, as we shall see a little later.

Consider the following minimization problem with ‘ $\geq$ ’-type constraints:

$$\text{Minimize } f(x) = 6x_1 + 10x_2,$$

subject to

$$\begin{aligned} x_1 + x_2 &\geq 2, \\ 2x_2 &\geq 3, \\ x_1, x_2 &\geq 0. \end{aligned} \tag{A.18}$$

We construct the following LP problem in the standard maximization format:

$$\text{Maximize } f(x) = -6x_1 - 10x_2,$$

subject to

$$\begin{aligned} -x_1 - x_2 + x_3 &\leq -2, \\ -2x_2 + x_4 &\leq -3, \\ x_1, x_2, x_3, x_4 &\geq 0. \end{aligned} \tag{A.19}$$

It is clear that the initial basic variables  $x_3$  and  $x_4$  both take negative values ( $x_3 = -2$  and  $x_4 = -3$ ). This is not allowed by the original simplex method and we cannot proceed with the simplex algorithm. One way out is to introduce artificial variables as discussed in Section A.3 and resort to a dual phase LP method.

Here, we suggest a more efficient algorithm. In the dual-simplex algorithm, the dual problem of the given primal problem is kept in mind. If

the dual problem is feasible, a method can be devised in principle to solve the dual problem, however, by actually solving the primal problem. As we have discussed earlier, at the optimum of the primal problem, the dual problem is also optimum. Thus, despite searching around the infeasible regions of the primal problem, an attempt to solve the dual problem should eventually take the algorithm to converge to the primal solution. We describe the dual-simplex algorithm in the following.

### Algorithm

**Step 1** Choose a dual-feasible solution, that is, for minimization problems, choose  $\mathbf{c} \geq 0$  and for maximization problems  $\mathbf{c} \leq 0$ . Set all nonbasic variables to zero. Set iteration counter  $t = 0$ .

**Step 2** If  $\mathbf{x}_B \geq 0$ , terminate. Else, go to Step 3.

**Step 3** Identify the leaving basic variable by the following criterion. Determine

$$x_r = \min_j \{x_j | x_j < 0\}.$$

That is, find the basic variable  $x_j$  that has the most negative  $b_j$  value. This variable  $x_r$  leaves the basic variable set.

**Step 4** Identify the nonbasic variable that should be made basic using the following criterion. Determine

$$q = \operatorname{argmin}_k \left\{ \left| \frac{(\Delta f)_k}{a_{rk}} \right| \middle| a_{rk} < 0 \right\}.$$

The nonbasic variable  $x_q$  replaces basic variable  $x_r$  in the simplex. If there is no  $a_{rk} < 0$ , there is no feasible solution to the problem.

**Step 5** Rewrite the simplex, perform a row-echelon formulation for new basic variables, set  $t = t + 1$ , and go to Step 2.

The reasons for operations in Steps 2 and 3 are given here. Note that  $-\mathbf{b}$  in the primal problem is equal to  $(\Delta f)_q$  of the dual problem. Since the variable having the highest positive  $(\Delta f)_q$  is chosen as a basic variable in the original simplex algorithm (in our dual problem), this amounts to choosing the variable having the most negative  $b_j$  value in the primal problem. The basic variables in the primal problem are equivalent to the nonbasic variables in the dual problem. Similarly the minimum ratio rule for choosing the nonbasic variable that would become basic in the original simplex algorithm (our dual problem) should now be applied as a ratio between  $(\Delta f)_k$  and  $a_{rk}$ , but only for those for which  $a_{rk}$  is negative. Thus, the above operations are designed in visualizing how the dual problem would have been solved by the original simplex method. Hence, this method is named the dual-simplex method.

We now illustrate the working of the dual-simplex algorithm on the example problem given in Equation A.19.

**Step 1** The problem is a maximization problem. Since both  $c_1$  and  $c_2$  are negative, it corresponds to a dual-feasible solution. Set  $t = 0$ . We present the initial tableau in the following:

		-6 -10 0 0				
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
0	$x_3$	-1	-1	1	0	-2
0	$x_4$	0	<span style="border: 1px solid black; padding: 2px;">-2</span>	0	1	-3 ←
$(\Delta f)_q$		-6	-10	0	0	$f(x) = 0$

↑

**Step 2** Here  $\mathbf{x}_B = (-2, -3)^T$ . Since they are negative, we do not terminate.

**Step 3** We now identify  $x_r$  by looking at all negative  $b_j$  values and choosing the most negative one. The tableau above indicates that  $x_r = x_4$  having  $b_4 = -3$ .

**Step 4** Now we identify the corresponding nonbasic variable that has  $a_{4k} < 0$  and minimum ratio between  $(\Delta f)_k$  and  $a_{4k}$ . In the above tableau, there is only one entry for which  $a_{4k}$  is negative. It is for variable  $x_2$ . That is,  $q = 2$ . This indicates that variable  $x_4$  should now be replaced by variable  $x_2$  in the next simplex.

We now form the second tableau and perform a row-echelon operation.

		-6 -10 0 0				
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
0	$x_3$	<span style="border: 1px solid black; padding: 2px;">-1</span>	0	1	-0.5	-0.5 ←
-10	$x_2$	0	1	0	-0.5	1.5
$(\Delta f)_q$		0	0	0	-5	$f(x) = -15$

↑

Set  $t = 1$  and go to Step 2.

**Step 2** At this point, the basic variables are  $(x_2, x_3) = (1.5, -0.5)$ . Since  $x_3 = -0.5$ , we do not terminate the algorithm.

**Step 3** We notice that  $x_r = x_3$ .

**Step 4** We also observe that  $q = 1$ . Thus, variable  $x_3$  will now get replaced by  $x_1$ .

**Step 4** Next, we form the third tableau.

		-6	-10	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
-6	$x_1$	1	0	-1	0.5	0.5
-10	$x_2$	0	1	0	-0.5	1.5
$(\Delta f)_q$				0	0	-6 -2
				$f(x) = -18$		

Set  $t = 2$  and go to Step 2.

**Step 2** At this point, the basic variable values are  $(x_1, x_2) = (0.5, 1.5)$  with a function value of -18. Now that both variables have non-negative values, we terminate the algorithm and declare  $(0.5, 1.5)^T$  as the optimal solution to the problem given in Equation (A.19).

Thus, the solution to the original minimization problem given in Equation (A.18) is  $(0.5, 1.5)^T$  with a function value of 18.

Recall that the problem given in Equation (A.19) is the dual version of the primal problem given in Equation (A.3). Thus, it is not surprising that both problems have the same optimal function values. Recall that in Section (A.8), we found that the dual solution to the same primal problem graphically to be  $(0.5, 1.5)^T$ . With the help of the dual-simplex method, we are now able to solve the dual problem and find an identical solution.

We make another observation from the above final tableau. The term  $\mathbf{c}_B^T \mathbf{B}^{-1}$  can be obtained from the  $(\Delta f)_q$  row and  $\mathbf{x}_S$  column of the tableau and is found to be  $(-6, -2)^T$ . Since this is supposed to be the optimal dual solution of this problem, it is not surprising that this solution is the solution to the original problem given in Equation (A.3). These calculations imply that in the case of linear programming problems,

$$\text{Dual of (Dual of } P) \equiv \text{Primal } P.$$

### A.9.1 Dual-Simplex Method for Sensitivity Analysis

The dual-simplex method is also useful in the sensitivity analysis after an LP has been solved using the primal-simplex method. We shall illustrate a couple of different sensitivity analysis procedures here.

#### Inclusion of additional constraints

Let us consider the primal problem given in Equation (A.3) again. After we solve the problem, we obtain the tableau shown in Table A.2. At this point, the optimal solution is  $(6, 2)^T$  with a function value of 18. To perform a sensitivity analysis, let us say that we would like to add a new constraint  $x_1 + x_2 \geq 2$  (as used in Section A.3) and resolve the problem, not by formulating a new problem, but starting from the final tableau obtained

after solving the original problem. With the third constraint, one more slack variable ( $x_5$ ) is needed:

$$-x_1 - x_2 + x_5 = -2.$$

The tableau becomes as follows:

		2	3	0	0	0
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
2	$x_1$	1	0	1	0	0
3	$x_2$	0	1	-0.5	0.5	0
0	$x_5$	-1	-1	0	0	1
$(\Delta f)_q$		0	0	-0.5	-1.5	0
						$f(x) = 18$

We shall now apply the dual-simplex method to find the new solution. Since the addition of a new constraint disturbs the tableau in terms of the rows being not in row-echelon form, we fix the tableau to make the rows in row-echelon form. For this purpose, first we add rows 1, 2 and 3 together and replace row 3 with the result. We obtain the following tableau:

		2	3	0	0	0
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
2	$x_1$	1	0	1	0	0
3	$x_2$	0	1	-0.5	0.5	0
0	$x_5$	0	0	0.5	0.5	1
$(\Delta f)_q$		0	0	-0.5	-1.5	0
						$f(x) = 18$

Since all the basic variables are non-negative, we terminate the dual-simplex procedure in Step 2 of the algorithm and declare  $(x_1, x_2) = (6, 2)$  as the optimal solution to the new problem. Notice that the addition of the new constraint  $x_1 + x_2 \geq 2$  does not alter the optimal solution and how quickly the dual-simplex method is able to find this fact. Recall the dual phase method solved the same problem in Section A.3 in a much more computationally expensive manner.

Let us now add a constraint  $x_2 \geq 3$  to the original problem. This changes the optimal solution to  $(4, 3)^T$  with a function value 17. Let us apply the dual-simplex method to investigate if it is able to find this solution. We shall begin with the final tableau obtained for solving the original problem. The tableau is modified with the new constraint in the following:

		2	3	0	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
2	$x_1$	1	0	1	0	0	6
3	$x_2$	0	1	-0.5	0.5	0	2
0	$x_5$	0	-1	0	0	1	-3
$(\Delta f)_q$		0	0	-0.5	-1.5	0	$f(x) = 18$

Since this disturbs the row-echelon nature of the rows, we add rows 2 and 3 together and replace row 3 with the result:

		2	3	0	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
2	$x_1$	1	0	1	0	0	6
3	$x_2$	0	1	-0.5	0.5	0	2
0	$x_5$	0	0	-0.5	0.5	1	-1
$(\Delta f)_q$		0	0	-0.5	-1.5	0	$f(x) = 18$

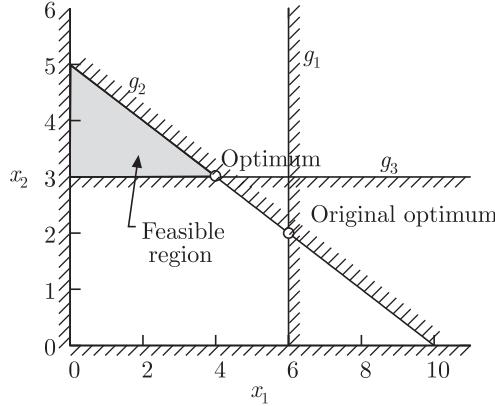
↑

Here,  $x_r = x_5$  and  $x_q = x_3$ . We now modify the tableau to make  $x_3$  to replace  $x_5$ , as follows:

		2	3	0	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
2	$x_1$	1	0	0	1	2	4
3	$x_2$	0	1	0	0	-1	3
0	$x_3$	0	0	1	-1	-2	2
$(\Delta f)_q$		0	0	0	-2	-1	$f(x) = 17$

Since all the basic variables have a positive value, we stop the dual-simplex algorithm. The optimal solution obtained from the final tableau is  $(4, 3)^T$  with a function value of 17. As an additional information, the corresponding shadow price (or, Lagrange multiplier) values are 0, 2 and 1 for constraints 1, 2 and 3, respectively. This means that the first constraint is not active at the current optimal solution. As shown in Figure A.7, the optimal solution corresponds to the intersection of the second and third constraints. Constraint 1 is inactive at the optimal solution.

Without solving the problem with old and new constraints together, the above dual-simplex method shows how the final simplex tableau of the original problem can be modified to take care of any new constraint to obtain the new optimal solution in a computationally quick manner.



**Figure A.7** A new constraint  $(x_2 \geq 3)$  is added. The optimum changes to  $(4, 3)^T$ .

### Changing RHS coefficients of constraints

Next, we discuss how the dual-simplex method can be used to perform sensitivity analysis for a change in the right-hand side coefficient of constraints. Note that the RHS coefficients cause the optimal solution to be  $\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b}$ . A change in the RHS coefficients does not change any other entities in the tableau. Thus, for a new  $\mathbf{b}$ -vector, we recompute  $\bar{\mathbf{b}} = \mathbf{B}^{-1}\mathbf{b}$  and replace the RHS vector in the tableau with  $\bar{\mathbf{b}}$ .

We illustrate the procedure here. Let us say that after solving the original problem given in Equation (A.3), we would like to change the RHS coefficient of the first constraint to 12 from 6, so that the constraint becomes  $x_1 \leq 12$ , and  $\mathbf{b} = (12, 10)^T$ . Note that we obtain the  $\mathbf{B}^{-1}$  matrix from the final tableau of the original problem as

$$\mathbf{B}^{-1} = \begin{bmatrix} 1 & 0 \\ -0.5 & 0.5 \end{bmatrix}.$$

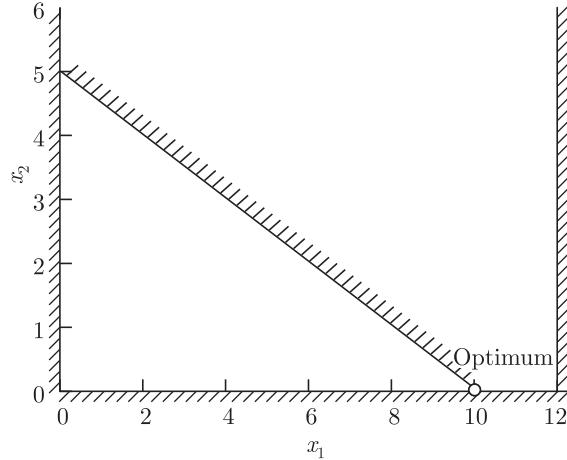
Thus, the new RHS vector is  $\mathbf{B}^{-1}\mathbf{b} = (12, -1)^T$ . We modify the final tableau as follows:

		2	3	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
2	$x_1$	1	0	1	0	12
3	$x_2$	0	1	-0.5	0.5	-1
$(\Delta f)_q$		0	0	-0.5	-1.5	

Since  $x_2 = -1$  (negative), we continue with the dual-simplex method. The dual-simplex operations indicate that  $x_r = x_2$  and  $x_q = x_3$ . We revise the tableau as follows:

		2	3	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	
2	$x_1$	1	2	0	1	10
3	$x_3$	0	-2	1	-1	2
$(\Delta f)_q$		0	-1	0	-2	$f(x) = 20$

Since non-negative basic variables are obtained, we stop the algorithm and declare  $(x_1, x_2) = (10, 0)^T$  as the optimal solution. The variable  $x_3$  being a slack variable, we do not care about its value much. Figure A.8 shows the feasible region and also indicates the same optimal solution. The first



**Figure A.8** RHS of  $g_1$  is changed to 12. The optimum changes to  $(10, 0)^T$ .

constraint is inactive, which is also obtained from the zero value of the Lagrange multiplier of the first constraint, as computed from the tableau:  $\mathbf{c}_B^T \mathbf{B}^{-1} = (0, 2)^T$ .

We now consider a final problem in which a new constraint  $x_1 + 5x_2 \geq 15$  is added to the original problem and the RHS of the first constraint is changed as  $x_1 \leq 12$ . The modified optimization problem is given as follows:

$$\text{Maximize } f(x) = 2x_1 + 3x_2$$

subject to

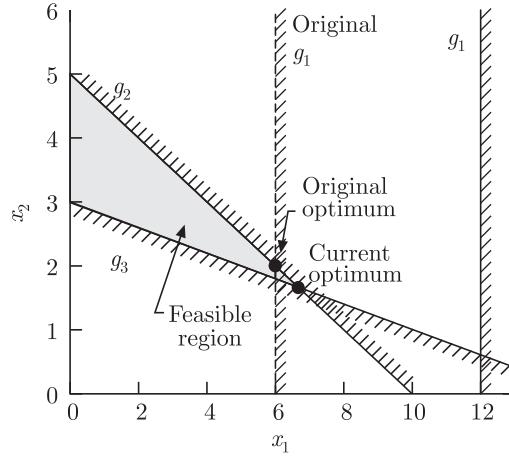
$$x_1 \leq 12,$$

$$x_1 + 2x_2 \leq 10,$$

$$x_1 + 5x_2 \geq 15,$$

$$x_1, x_2 \geq 0.$$

Figure A.9 shows the feasible region and the optimal solution. It can be seen that the optimal solution is  $(6, 67, 1.67)^T$  having a function value equal to 18.33.



**Figure A.9** A new constraint ( $g_3$ ) is added and RHS of  $g_1$  is changed. Dual-simplex method finds the true optimum  $(6.67, 1.67)^T$ .

We begin solving the problem from the final tableau of the original simplex method (of solving the problem in Equation (A.3)) by adding a third row corresponding to the new constraint and by changing the RHS vector for the first two rows as  $\mathbf{B}^{-1}\mathbf{b} = (12, -1)^T$ . The modified tableau is as follows:

		2	3	0	0	0
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
2	$x_1$	1	0	1	0	0
3	$x_2$	0	1	-0.5	0.5	0
0	$x_5$	-1	-5	0	0	1
$(\Delta f)_q$		0	0	-0.5	-1.5	0

We perform a row-echelon operation and obtain the following tableau.

		2	3	0	0	0
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
2	$x_1$	1	0	1	0	0
3	$x_2$	0	1	-0.5	0.5	0
0	$x_5$	0	0	-1.5	2.5	1
$(\Delta f)_q$		0	0	-0.5	-1.5	0

↑ ←

The dual-simplex method identifies  $x_r = x_5$  and  $x_q = x_3$ . We update the tableau as follows:

		2	3	0	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
2	$x_1$	1	0	0	1.67	0.67	6.67
3	$x_2$	0	1	0	-0.33	-0.33	1.67
0	$x_3$	0	0	1	-1.67	-0.67	5.33
$(\Delta f)_q$		0	0	0	-2.33	-0.33	$f(x) = 18.33$

All basic variables take non-negative values, hence the tableau represents the optimal solution  $(x_1, x_2) = (6.67, 1.67)^T$  with a function value equal to 18.33. As a by-product, we obtain the shadow prices as zero, 2.33 and 0.33 for constraints 1, 2, and 3, respectively. As evident, the first constraint is inactive and the optimal solution lies at the intersection of constraints 2 and 3.

## A.10 Summary

Linear programming methods are widely used in problems where the objective function as well as the constraints are linear. Usually, all design variables are restricted to be nonnegative. In these problems, one of the corner points of the feasible search space is the optimum point. The simplex method of linear programming begins from a basic feasible solution (a corner point in the feasible search space) and moves to a neighbouring basic feasible solution that increases the objective function value the most. This feature of the simplex search method makes it efficient and popular in solving various linear programming problems. In such formulations, inequality constraints are converted into equality constraints by adding or subtracting slack variables. Often, the addition of slack variables alone cannot find an initial basic feasible solution. Artificial variables are then added and a dual phase strategy is used to first find an initial basic feasible solution and then the regular simplex method is used to find the exact optimal solution. Simple numerical exercise problems are taken to illustrate the working of the simplex method. Interested readers may refer to other textbooks for more details on linear programming methods (Taha, 1989).

Thereafter, the Big-M method is described to handle greater-than-equal-to type inequality constraints. An algebraic description of simplex method is presented next to provide a symbolic representation of the working principle of the simplex method. Initial and final simplexes are shown in algebraic form, so that a clear idea of the obtained quantities at the final simplex can be obtained. The results are used in subsequent sections related to sensitivity analysis and the dual-simplex method.

The simplex method can be implemented in a computationally efficient manner. This is done in the next section entitled ‘Revised Simplex Method’. An example problem is also shown to illustrate the working of the revised simplex method. Two types of sensitivity analyses for LP problems are discussed next—sensitivity with respect to a change in the right-hand side value of constraints and sensitivity with respect to a change in the cost terms. Thereafter, duality theory related to LP problems are described. Finally, the famous dual-simplex method is presented and its use in performing both sensitivity analyses is illustrated with examples.

## REFERENCES

- Dantzig, G. B. (1963). *Linear Programming and Extensions*. Princeton: Princeton University Press.
- Lundgren, J., Rönnqvist, M., Värbrand, P. and Henningsson, M. (2011). Optimization with exercises. New Delhi: Overseas Press.
- Murty, K. (1983). *Linear Programming*. New York: Wiley.
- Strang, G. (1980). *Linear Algebra and Its Applications*. Orlando: Academic Press.
- Taha, H. A. (1989). *Operations Research*. New York: Macmillan.

## PROBLEMS

**A-1** Solve the following problems using the simplex method:

(a) Maximize  $x_1 + x_2$

subject to

$$3x_1 + 5x_2 \leq 6,$$

$$0 \leq x_1, x_2 \leq 1.$$

(b) Maximize  $x_1$

subject to

$$2x_1 + x_2 \leq 2,$$

$$x_1 + 5x_2 + 10 \geq 0,$$

$$x_2 \leq 1.$$

(c) Maximize  $x_1 + 2x_2$

subject to

$$x_1 + 2x_2 + 5x_3 \geq 3,$$

$$x_1 + 2x_2 + 5x_3 \leq 10,$$

$$x_1, x_2, x_3 \geq 0.$$

(d) Minimize  $8x_1 + 3x_2$

subject to

$$8x_1 - 3x_2 + 2x_3 \geq 2,$$

$$4x_1 + 2x_2 + x_3 \leq 5,$$

$$x_1, x_2, x_3 \geq 0$$

(e) Minimize  $3x_1 + x_2$

subject to

$$4x_1 + x_2 \geq 3,$$

$$4x_1 + 3x_2 \leq 6,$$

$$x_1 + 2x_2 \leq 3,$$

$$x_1, x_2 \geq 0.$$

**A-2** Solve the following linear program by using the simplex method (tableau method):

$$\text{Maximize } 0.9x_1 + x_2,$$

subject to

$$2x_1 + 3x_2 \leq 9,$$

$$|x_2 - x_1| \leq 1,$$

$$x_1, x_2 \geq 0.$$

Plot a neat sketch of the constraints and the feasible region and mark the proceedings of each tableau on the plot.

**A-3** Solve the following linear program by using the simplex method (tableau method):

$$\text{Maximize } x_1 + x_2,$$

subject to

$$2x_1 + 3x_2 \leq 12,$$

$$|x_2 - x_1| \leq 1,$$

$$x_1, x_2 \geq 0.$$

Plot a neat sketch of the constraints and the feasible region and mark the proceedings of each tableau on the plot.

**A-4** Solve the following LP problem using simplex method:

$$\text{Maximize } 5x_1 + 2x_2 + x_3$$

subject to

$$4x_1 + 2x_2 + x_3 \leq 8,$$

$$3x_1 + 2x_2 + 2x_3 \leq 10,$$

$$x_1 + x_2 + x_3 \geq 4,$$

$$x_1, x_2, x_3 \geq 0,$$

**A-5** Find the intersecting point of the following pairs of straight lines by formulating an LP program:

$$(a) \quad x_1 - 2x_2 + 1 = 0,$$

$$5x_1 + 3x_2 - 10 = 0.$$

$$(b) \quad x_1 + x_2 - 1 = 0,$$

$$10x_1 + x_2 + 5 = 0.$$

**A-6** Consider the following optimization problem that finds a search direction which is descent as well as feasible at the point  $x^{(t)}$ :

$$\text{Maximize } \theta$$

subject to

$$\nabla f(x^{(t)}) \cdot t \leq \nabla f(x^{(t)}) \cdot A - \theta,$$

$$\nabla g_j(x^{(t)}) \cdot t \geq \nabla g_j(x^{(t)}) \cdot A + \theta, \quad j = 1, 2, \dots, J,$$

where  $A$  is an  $N$ -dimensional vector of ones. Find the optimal values of  $t_i$ ,  $i = 1, 2, \dots, N$ , and  $\theta$  for the following optimization problems using the simplex method.

- (i)  $f(x) = x^3 - x$ ,  $g(x) = x \geq 0.2$ ,  $x^{(t)} = 0.2$ .
- (ii)  $f(x) = x^3 - x$ ,  $g(x) = x \leq 0.8$ ,  $x^{(t)} = 0.8$ .
- (iii)  $f(x) = 2x_1^2 + x_2^2$ ,  $g(x) = x_1^3 - 2x_2 \geq 19$ ,  $x^{(t)} = (3, 4)$ .
- (iv)  $f(x) = 2x_1^2 x_2 - x_1$ ,  $g_1(x) = 2x_1 + x_2 \geq 5$ ,  $g_2(x) = x_1^2 \leq 16$ ,  
 $x^{(t)} = (3, 1)$ .

**A-7** Solve the following problem using the Big-M method:

$$\text{Maximize } 4x_1 + 3x_2$$

subject to

$$10x_1 + 3x_2 \leq 30,$$

$$x_2 \leq 2,$$

$$2x_1 + x_2 \geq 1,$$

$$x_1, x_2 \geq 0.$$

**A-8** Solve the following problem using the Big-M method:

$$\text{Maximize } 3x_1 + 2x_2$$

subject to

$$3x_1 + 8x_2 \leq 24,$$

$$7x_1 + 2x_2 \leq 14,$$

$$2x_1 + x_2 \geq 1,$$

$$x_1 + 2x_2 \geq 1,$$

$$x_1, x_2 \geq 0.$$

**A-9** Solve Problem A.3 using the revised simplex method.

**A-10** Solve the following problem using the dual-simplex method:

$$\text{Maximize } x_1 + x_2$$

subject to

$$3x_1 + 5x_2 \leq 7,$$

$$0 \leq x_1, x_2 \leq 1.$$

**A-11** Formulate the dual problem and solve to find the dual solution:

$$\text{Maximize } 3x_1 + 2x_2$$

subject to

$$x_1 + 3x_2 \leq 3,$$

$$5x_1 - x_2 = 4,$$

$$x_1 + 2x_2 \leq 2,$$

$$x_1, x_2 \geq 0.$$

**A-12** For the following problem:

$$\text{Maximize } 2x_1 + 3x_2 + 4x_3$$

subject to

$$x_1 + 2x_2 + x_3 \leq 4,$$

$$2x_1 + 3x_2 + 3x_3 \geq 1,$$

$$x_1 + 2x_2 \leq 2,$$

$$x_1, x_2 \geq 0, \quad x_3 \text{ free},$$

- (i) Construct the dual problem,
- (ii) Choose an arbitrary feasible solution to the primal problem and dual problem and show that the weak duality theorem is satisfied.
- (iii) Determine the optimal solutions of the primal and dual problems.

**A-13** Find the range of values of  $b_1$  and  $b_2$  for which the optimum of the problem remains feasible:

$$\text{Maximize } x_1 + x_2$$

subject to

$$2x_1 + 7x_2 \leq b_1 = 14,$$

$$5x_1 + 3x_2 \leq b_2 = 15,$$

$$x_1, x_2 \geq 0.$$

**A-14** Find the range of values of the cost terms in the above LP Problem A.13 for which the optimum solution does not change.

**A-15** If the objective function of Problem A.13 is changed to  $1.5x_1 + 0.8x_2$ , determine if the original optimal solution remains as optimal using 100% rule.

**A-16** If a new constraint  $3x_1 + 5x_2 \leq 15$  is added to Problem A.13, determine what would be the new optimal solution using the dual-simplex method.

**A-17** Consider the following LP problem:

$$\text{Maximize } x_1 + 3x_2$$

subject to

$$x_1 + x_2 \leq 8,$$

$$-x_1 + x_2 \leq 4,$$

$$x_1 \leq 6,$$

$$x_1, x_2 \geq 0.$$

Determine graphically and using the simplex tableau the following:

- (i) the optimal solution,
- (ii) the shadow price of each of the three constraints,
- (iii) the range of cost coefficients for  $x_1$  and  $x_2$  (changed one at a time) for which the optimal solution of the original LP does not change,
- (iv) the range of right-hand side coefficients for  $x_1$  and  $x_2$  (changed one at a time) for which the optimal solution of the original LP remains feasible.

**A-18** Find the shadow prices of the following problem:

$$\text{Maximize } x_1 + x_2$$

subject to

$$3x_1 + 5x_2 \leq 7,$$

$$0 \leq x_1, x_2 \leq 1.$$

**A-19** Consider the following LP problem:

$$\text{Maximize } x_1 + 2x_2$$

subject to

$$2x_1 + x_2 \leq 4,$$

$$x_1 - x_2 \leq 2,$$

$$x_1, x_2 \geq 0.$$

- (i) Find the optimal solution using the simplex method.
- (ii) Using the final tableau, determine the new optimal solution if the right-hand side coefficient of the first constraint is changed to 5.
- (iii) Determine the new optimal solution if the right-hand side coefficients of the first and second constraints are changed to 5 and 1.8, respectively. Use the 100% rule to determine if the optimal solution remains feasible under the above changes.
- (iv) Determine the new optimal solution, when a new constraint  $x_2 \leq 1.5$  is added.
- (v) Determine the optimal solution, when a new constraint  $x_2 \leq 1.5$  is added and the right-hand side coefficient of the first constraint is changed to 5.

**A-20** For the LP problem given below:

$$\text{Maximize } 3x_1 + 2x_2$$

subject to

$$2x_1 + x_2 \leq 10,$$

$$x_1 + x_2 \leq 8,$$

$$x_1 \leq 4,$$

$$x_1, x_2 \geq 0.$$

the following final tableau is obtained ( $x_3$ ,  $x_4$  and  $x_6$  are slack variables for three constraints, respectively):

		2	3	0	0	0	
$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	
2	$x_2$	0	1	-1	2	0	6
0	$x_5$	0	0	-1	1	1	2
3	$x_1$	1	0	1	-1	0	2
$(\Delta f)_q$		0	0	-10	-10	0	$f(\mathbf{x}) = 18$

- (i) Determine the optimal value of the dual variables.
- (ii) For which range of values of the objective function coefficients (one at a time), is the above solution optimal?
- (iii) For which values of the right-hand side coefficients (one at a time), is the above solution feasible?

**A-21** For the following LP problem:

$$\text{Minimize } 28x_1 + 30x_2 + 20x_3$$

subject to

$$4x_1 + x_2 + 5x_3 \leq 3.5,$$

$$4x_1 + 4x_2 + 3x_3 \geq 3,$$

$$x_1 + x_2 + x_3 \geq 1,$$

$$x_1, x_2, x_3 \geq 0,$$

the following optimal tableau is obtained ( $x_4$ ,  $x_5$ , and  $x_6$  are slack variables for three constraints):

$c_B$	Basic	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	
0	$x_5$	-0.75	0	0	-0.25	1	-4.25	0.375
-20	$x_3$	0.75	0	1	0.25	0	0.25	0.625
-30	$x_2$	0.25	1	0	-0.25	0	-1.25	0.375
$(\Delta f)_q$		-5.5	0	0	-2.5	0	-32.5	$f(\mathbf{x}) = -23.75$

- (i) Determine the optimal value of the dual variables.
- (ii) For which values of the objective function coefficients (one at a time), is the above solution optimal?
- (iii) For which values of the right-hand side coefficients (one at a time), is the above solution feasible and optimal?

# Index

---

- 100% rule for cost coefficients, 393
- 100% rule for RHS coefficients, 391
- Active constraint, 144
  - strategy, 232, 239
- Ant-based EMO, 323
- Arithmetic-geometric-mean inequality, 278
- Artificial neural networks, 18
- Artificial variables, 377
- Basic feasible solution, 371, 377
- Basic variable, 371
- BFS method, 134
- Bi-level optimization, 38
- Big-M method, 381
- Binary tournament selection, 314
- Bisection method, 65
  - algorithm of, 65
- BLX operator, 316
- Boltzmann distribution, 325
- Bounding phase method, 49, 73, 88, 106
  - algorithm of, 49
  - computer code of, 78
- Box's evolutionary optimization, 90
  - algorithm of, 90
- Box's method
  - constrained functions for, 177
  - unconstrained functions for, 90
- Bracket operator penalty, 156, 162
- Bracketing algorithms, 46
- Branch-and-bound method, 270
  - algorithm of, 270
- Car suspension design, 12
  - dynamic model of, 12
- Cauchy's method, 112–114, 118, 204
  - algorithm of, 112
- Central difference technique, 108
- Classical optimization methods, 37
- Classification of optimization methods, 35
- Complex search method, 177–182
  - algorithm of, 178
- Conditioned simplex method, 215
- Conjugate, 103
- Conjugate direction method, 103–108
  - algorithm of, 105
- Conjugate gradient method, 120–124
  - algorithm of, 121
- Constrained domination, 323
- Constrained optimization algorithms, 143
- Constraint qualification, 149
- Constraints, 4
  - active, 144, 233
  - equality, 4, 143
  - inactive, 144
  - inequality, 4, 144

- Contour plot, 87
- Contraction, 95
- Control systems, 22
- Convergence, 113
- Convex function, 150
- Crane maneuvering, 22
- Crossover, 296
  - single-point, 296, 306
  - two-point, 315
  - uniform, 315
- Crowding distance, 321
- Cubic search method, 68
  - algorithm of, 69
- Customized optimization method, 39
- Cutting plane method, 192–202, 369
  - algorithm of, 193
  - cut-deletion, 198
    - algorithm of, 198
  - nonlinear objective function, 201
- Data fitting, 21
- Data mining, 31
- Decision variables, 3
- Degree of difficulty, 279
- Derivatives
  - multivariable for, 108
  - single-variable for, 64
- Descent direction, 109, 112
  - steepest, 111
- Design and manufacturing problems, 9
- Design variables, 3
  - artificial, 29
- Deterministic optimization, 39
- DFP method, 124–128
  - algorithm of, 125
  - differential evolution, 323
- Direct search methods
  - constrained, 173
  - multivariable, 89
  - single-variable, 43
- Discrete-variable problems, 269
- Distribution index, 316
- Dual phase method, 208, 377, 398, 402
- Dual problem, 151, 278, 395
- Dual variables, 278
- Dual-simplex method, 398
  - algorithm of, 399
- Duality gap, 151
- Duality principle, 6, 43, 144
- Duality theory, 151, 394
- Dynamic optimization, 39
- Dynamic programming method, 37
- Enumerative integer programming method, 264
- Error minimization, 91
- Exhaustive search method, 46
  - algorithm of, 47
- Expansion, 95
- Exploratory move, 98, 100
- Extended parallel subspace property, 104
- Feasible direction, 203
- Feasible direction method, 203–212
  - algorithm of, 203
- Feasible point, 144
- Fibonacci numbers, 54
- Fibonacci search method, 54
  - algorithm of, 55
- Finite element software, 12
- Fletcher-Reeves method, 121–124
  - algorithm of, 121
- Formulation, 3
  - engineering problems of, 8
- Frank-Wolfe method, 186–192, 369
  - algorithm of, 186
- Fuzzy rule-base, 34
- GAs (*see* Genetic algorithms)
- Gauss-elimination method, 279
- Gaussian distribution, 327
- Generalized geometric programming, 284
- Genetic algorithms (GAs), 37, 292–325
  - algorithm of, 302
  - children strings in, 296
  - coding in, 293
  - computer code, 348
  - constrained problems, 311
  - differences with other methods, 298
  - engineering applications with, 325
  - fitness function in, 294
  - mating pool in, 296
  - multimodal functions for, 323

- multiobjective functions for, 319  
operators, 294  
crossover, 296  
mutation, 297  
reproduction, 294  
parent strings in, 296  
real-coded, 315  
schema in, 298  
similarities with other methods, 301  
working principles of, 293
- Geometric programming (GP), 278–287  
algorithm of, 282
- Global optimization, 330  
genetic algorithms, 332  
simulated annealing, 334  
steepest descent method, 330
- Global optimum, 44, 330
- Golden number, 57
- Golden section search, 57, 63, 73, 88, 106  
algorithm of, 58  
computer code of, 80
- GP method (*see* Geometric programming)
- Gradient (*see* Derivatives)
- Gradient projection method, 232–239  
algorithm of, 233
- Gradient-based methods  
constrained, 186  
multivariable, 108  
single-variable, 63
- GRG method (*see* Reduced gradient method)
- Headway, 29
- Hessian matrix, 86, 114, 120, 125
- Heuristic optimization, 40
- Himmelblau's function  
constrained, 146, 194  
unconstrained, 91, 110
- Hooke-Jeeves method, 98–103  
algorithm of, 99
- Hypercube, 90
- Inactive constraint, 144
- Infeasible point, 144
- Infinite barrier penalty, 155
- Inflection point, 44, 86
- Integer programming, 264  
branch-and-bound method, 270–277  
penalty function method, 265–270
- Intelligent system design, 32
- Interval halving method, 52  
algorithm of, 52
- Inverse penalty, 155
- Inverse problems, 24
- Jerk, 14
- Job shop scheduling, 325
- K-T points (*see* Kuhn-Tucker points)
- KKT optimality conditions, 144, 212
- Knowledge base, 33
- Kuhn-Tucker, 144–150  
conditions of, 39, 144  
necessity theorem, 148  
points, 144  
sufficiency theorem, 150
- KUR, 322
- Lagrange multipliers, 145, 169, 281
- Lagrangian dual problem, 151
- Lagrangian duality theory, 151
- Level of service (LOS), 26
- Line search, 113
- Linear programming (LP), 369  
algebraic description, 382  
big-M method, 381  
dual-phase method, 377  
dual-simplex method, 398  
duality theory, 394  
revised simplex method, 385  
sensitivity analysis, 388  
shadow price, 384  
simplex method, 186, 206, 371–381  
algorithm of, 373
- Linearly independent directions, 103, 122
- Local optimum, 44, 161
- Local search algorithm, 37
- Log penalty, 155
- LP method (*see* Linear programming (LP))
- Luus-Jaakola method, 182–185  
algorithm of, 182

- Marquardt's method, 118–120
  - algorithm of, 118
- Mating pool, 307
- Metal cutting optimization problem, 15
- Method of multiplier, 162
  - algorithm of, 163
- Metropolis algorithm, 325, 334
- Micro-GA, 323
- Minimum ratio rule, 372
- Modeling problems, 17
- MOM (*see* Method of multiplier), 169
- MOMGA, 323
- Multimodal optimization, 38, 330
  - simultaneous solutions of, 324
- Multiobjective optimization, 38, 319
- Multivariable optimization algorithms, 85
- Mutation, 297, 307
  
- Negative-definite matrix, 86
- Nelder and Mead method, 95
  - algorithm of, 96
- Nested optimization, 38
- Newton's method, 114–118, 124
  - algorithm of, 115
- Newton-Raphson method, 63, 129, 176, 226
  - algorithm of, 63
- NLP problem (*see* Nonlinear programming problem)
- Non-stationary function optimization, 325
- Nonbasic variable, 371
- Nonlinear programming problem, 8
- Nontraditional optimization
  - algorithms, 292
  - methods, 37
- NSGA-II, 320
- Numerical derivatives
  - multivariable for, 108
  - single-variable for, 64
  
- Objective function, 5
- Off-line optimization, 33
- One-dimensional search, 87
- One-variable-at-a-time strategy, 90
- Optimal rule-base, 33
  
- Optimality criteria, 39, 85
  - constrained functions, 144
  - multivariable functions, 85
  - single-variable functions, 44
- Optimization procedure, 2
  - naive method, 2
  
- Parabolic penalty, 155
- Parallel subspace property, 103, 106
  - extended, 104
- Pareto-optimal solutions, 38
- Partial derivatives
  - multivariable for, 108, 115
  - single-variable for, 64
- Particle swarm EMO, 323
- Pattern move, 99, 101
- Pattern search method, 98
  - algorithm of, 99
- Penalty function method, 155, 312
  - algorithm of, 156
  - computer code for, 253
- Penalty terms, 155
  - bracket operator, 156
- Penalty type
  - exterior, 154
  - interior, 154
  - mixed, 154
- Polynomial mutation operator, 318
- Population-based search, 90, 95, 292
- Positive-definite matrix, 86
- Posynomial function, 278
- Powell's conjugate direction method, 103
  - algorithm of, 105
- Powell's method, 61
- Prediction, 32
- Primal problem, 151, 278, 395
- Probability of crossover, 306
- Probability of mutation, 297
  
- Quadratic estimation method, 60
  - algorithm of, 61
- Quadratic function, 103, 121
- Quadratic programming, 212
  
- Random number generator, 303

- Random numbers, 183, 328  
Random search method, 182–185  
    algorithm of, 182  
Reactor design, 18  
Reduced gradient (GRG) method, 224–232  
    algorithm of, 227  
Reflection, 95  
Region-elimination methods, 51  
Regression, 21  
Reproduction, 294  
    roulette-wheel selection, 295, 305  
    tournament selection, 314  
Restart, 116  
Revised simplex method, 385  
    algorithm of, 385  
Ride, 15  
    robot navigation, 33  
Root finding, 71, 176  
Routing problems, 325  
Row-echelon form, 207, 370, 374  
Runge-Kutta method, 14
- SBX, 316  
Scheduling optimization, 26  
Schema, 310  
Secant method, 67  
    algorithm of, 67  
Selection (*see* Reproduction)  
Selection pressure, 314  
Sensitivity analysis, 167, 388  
    dual-simplex method, 401  
    linear programming, 384  
Sequential quadratic programming (SQP) method, 218  
    algorithm of, 219  
Shadow price, 163, 384  
Simplex, 95  
Simplex search method, 95–98, 177  
    algorithm of, 96  
Simulated annealing method, 37, 325–329  
    algorithm of, 326
- Simulated binary crossover, 316  
Single-variable optimization algorithms, 43  
Slack variable strategy, 226, 228, 370  
Stationary point, 86  
Steepest descent direction, 110  
Steepest descent method, 112–114, 158, 330  
    algorithm of, 112  
    computer code of, 134  
Step-by-step format, 36  
Stochastic optimization, 39  
Stochastic programming method, 37  
Strong duality theorem, 396  
Strong duality theory, 152
- Test problems  
    KUR, 322  
Tournament selection, 323  
Traditional optimization methods, 37  
Transformation method, 154  
Transit scheduling, 26  
Transmissibility, 15  
Travelling salesperson problem, 325  
Truss structure design, 9, 341
- Unidirectional search, 87–89, 105
- Variable bounds, 6  
Variable elimination method, 173–177, 287  
Variable-metric method, 124
- Weak duality theorem, 151, 396  
    ZDT2, 321  
Zero-finding problem, 226  
Zoutendijk’s method, 203–212  
    algorithm of, 203