

0

Introduction

- 0.1. Objective and Approach
- 0.2. Organization of the Book
- 0.3. Examples
- 0.4. Programs
- 0.5. Problems
- 0.6. Significant Digits, Precision, Accuracy, Errors, and Number Representation
- 0.7. Software Packages and Libraries
- 0.8. The Taylor Series and the Taylor Polynomial

This Introduction contains a brief description of the objectives, approach, and organization of the book. The philosophy behind the Examples, Programs, and Problems is discussed. Several years' experience with the first edition of the book has identified several simple, but significant, concepts which are relevant throughout the book, but the place to include them is not clear. These concepts, which are presented in this Introduction, include the definitions of significant digits, precision, accuracy, and errors, and a discussion of number representation. A brief description of software packages and libraries is presented. Last, the Taylor series and the Taylor polynomial, which are indispensable in developing and understanding many numerical algorithms, are presented and discussed.

0.1 OBJECTIVE AND APPROACH

The objective of this book is to introduce the engineer and scientist to numerical methods which can be used to solve mathematical problems arising in engineering and science that cannot be solved by exact methods. With the general accessibility of high-speed digital computers, it is now possible to obtain rapid and accurate solutions to many complex problems that face the engineer and scientist.

The approach taken is as follows:

1. Introduce a type of problem.

2. Present sufficient background to understand the problem and possible methods of solution.
3. Develop one or more numerical methods for solving the problem.
4. Illustrate the numerical methods with examples.

In most cases, the numerical methods presented to solve a particular problem proceed from simple methods to complex methods, which in many cases parallels the chronological development of the methods. Some poor methods and some bad methods, as well as good methods, are presented for pedagogical reasons. Why one method does not work is almost as important as why another method does work.

0.2 ORGANIZATION OF THE BOOK

The material in the book is divided into three main parts:

- I. Basic Tools of Numerical Analysis
- II. Ordinary Differential Equations
- III. Partial Differential Equations

Part I considers many of the basic problems that arise in all branches of engineering and science. These problems include: solution of systems of linear algebraic equations, eigenproblems, solution of nonlinear equations, polynomial approximation and interpolation, numerical differentiation and difference formulas, and numerical integration. These topics are important both in their own right and as the foundation for Parts II and III.

Part II is devoted to the numerical solution of ordinary differential equations (ODEs). The general features of ODEs are discussed. The two classes of ODEs (i.e., initial-value ODEs and boundary-value ODEs) are introduced, and the two types of physical problems (i.e., propagation problems and equilibrium problems) are discussed. Numerous numerical methods for solving ODEs are presented.

Part III is devoted to the numerical solution of partial differential equations (PDEs). Some general features of PDEs are discussed. The three classes of PDEs (i.e., elliptic PDEs, parabolic PDEs, and hyperbolic PDEs) are introduced, and the two types of physical problems (i.e., equilibrium problems and propagation problems) are discussed. Several model PDEs are presented. Numerous numerical methods for solving the model PDEs are presented.

The material presented in this book is an introduction to numerical methods. Many practical problems can be solved by the methods presented here. Many other practical problems require other or more advanced numerical methods. Mastery of the material presented in this book will prepare engineers and scientists to solve many of their everyday problems, give them the insight to recognize when other methods are required, and give them the background to study other methods in other books and journals.

0.3 EXAMPLES

All of the numerical methods presented in this book are illustrated by applying them to solve an example problem. Each chapter has one or two example problems, which are solved by all of the methods presented in the chapter. This approach allows the analyst to compare various methods for the same problem, so accuracy, efficiency, robustness, and ease of application of the various methods can be evaluated.

Most of the example problems are rather simple and straightforward, thus allowing the special features of the various methods to be demonstrated clearly. All of the example problems have exact solutions, so the errors of the various methods can be compared. Each example problem begins with a reference to the problem to be solved, a description of the numerical method to be employed, details of the calculations for at least one application of the algorithm, and a summary of the remaining results. Some comments about the solution are presented at the end of the calculations in most cases.

0.4 PROGRAMS

Most numerical algorithms are generally expressed in the form of a computer program. This is especially true for algorithms that require a lot of computational effort and for algorithms that are applied many times. Several programming languages are available for preparing computer programs: FORTRAN, Basic, C, PASCAL, etc., and their variations, to name a few. Pseudocode, which is a set of instructions for implementing an algorithm expressed in conceptual form, is also quite popular. Pseudocode can be expressed in the detailed form of any specific programming language.

FORTRAN is one of the oldest programming languages. When carefully prepared, FORTRAN can approach pseudocode. Consequently, the programs presented in this book are written in simple FORTRAN. There are several vintages of FORTRAN: FORTRAN I, FORTRAN II, FORTRAN 66, 77, and 90. The programs presented in this book are compatible with FORTRAN 77 and 90.

Several programs are presented in each chapter for implementing the more prominent numerical algorithms presented in the chapter. Each program is applied to solve the example problem relevant to that chapter. The implementation of the numerical algorithm is contained within a completely self-contained *application subroutine* which can be used in other programs. These *application subroutines* are written as simply as possible so that conversion to other programming languages is as straightforward as possible. These subroutines can be used as they stand or easily modified for other applications.

Each *application subroutine* is accompanied by a *program main*. The variables employed in the *application subroutine* are defined by comment statements in *program main*. The numerical values of the variables are defined in *program main*, which then calls the *application subroutine* to solve the example problem and to print the solution. These main programs are not intended to be convertible to other programming languages. In some problems where a function of some type is part of the specification of the problem, that function is defined in a *function subprogram* which is called by the *application subroutine*.

FORTRAN compilers do not distinguish between uppercase and lowercase letters. FORTRAN programs are conventionally written in uppercase letters. However, in this book, all FORTRAN programs are written in lowercase letters.

0.5 PROBLEMS

Two types of problems are presented at the end of each chapter:

1. Exercise problems
2. Applied problems

Exercise problems are straightforward problems designed to give practice in the application of the numerical algorithms presented in each chapter. Exercise problems emphasize the mechanics of the methods.

Applied problems involve more applied engineering and scientific applications which require numerical solutions.

Many of the problems can be solved by hand calculation. A large number of the problems require a lot of computational effort. Those problems should be solved by writing a computer program to perform the calculations. Even in those cases, however, it is recommended that one or two passes through the algorithm be made by hand calculation to ensure that the analyst fully understands the details of the algorithm. These results also can be used to validate the computer program.

Answers to selected problems are presented in a section at the end of the book. All of the problems for which answers are given are denoted by an asterisk appearing with the corresponding problem number in the problem sections at the end of each chapter. The **Solutions Manual** contains the answers to nearly all of the problems.

0.6 SIGNIFICANT DIGITS, PRECISION, ACCURACY, ERRORS, AND NUMBER REPRESENTATION

Numerical calculations obviously involve the manipulation (i.e., addition, multiplication, etc.) of numbers. Numbers can be integers (e.g., 4, 17, -23, etc.), fractions (e.g., $1/2$, $-2/3$, etc.), or an infinite string of digits (e.g., $\pi = 3.1415926535\dots$). When dealing with numerical values and numerical calculations, there are several concepts that must be considered:

1. Significant digits
2. Precision and accuracy
3. Errors
4. Number representation

These concepts are discussed briefly in this section.

Significant Digits

The **significant digits**, or figures, in a number are the digits of the number which are known to be correct. Engineering and scientific calculations generally begin with a set of data having a known number of significant digits. When these numbers are processed through a numerical algorithm, it is important to be able to estimate how many significant digits are present in the final computed result.

Precision and Accuracy

Precision refers to how closely a number represents the number it is representing. **Accuracy** refers to how closely a number agrees with the true value of the number it is representing.

Precision is governed by the number of digits being carried in the numerical calculations. Accuracy is governed by the errors in the numerical approximation. Precision and accuracy are quantified by the errors in a numerical calculation.

Errors

The **accuracy** of a numerical calculation is quantified by the **error** of the calculation. Several types of errors can occur in numerical calculations.

1. Errors in the parameters of the problem (assumed nonexistent).
2. Algebraic errors in the calculations (assumed nonexistent).
3. Iteration errors.
4. Approximation errors.
5. Roundoff errors.

Iteration error is the error in an iterative method that approaches the exact solution of an exact problem asymptotically. Iteration errors must decrease toward zero as the iterative process progresses. The iteration error itself may be used to determine the successive approximations to the exact solution. Iteration errors can be reduced to the limit of the computing device. The errors in the solution of a system of linear algebraic equations by the successive-over-relaxation (SOR) method presented in Section 1.5 are examples of this type of error.

Approximation error is the difference between the exact solution of an exact problem and the exact solution of an approximation of the exact problem. Approximation error can be reduced only by choosing a more accurate approximation of the exact problem. The error in the approximation of a function by a polynomial, as described in Chapter 4, is an example of this type of error. The error in the solution of a differential equation where the exact derivatives are replaced by algebraic difference approximations, which have truncation errors, is another example of this type of error.

Roundoff error is the error caused by the finite word length employed in the calculations. Roundoff error is more significant when small differences between large numbers are calculated. Most computers have either 32 bit or 64 bit word length, corresponding to approximately 7 or 13 significant decimal digits, respectively. Some computers have extended precision capability, which increases the number of bits to 128. Care must be exercised to ensure that enough significant digits are maintained in numerical calculations so that roundoff is not significant.

Number Representation

Numbers are represented in number systems. Any number of bases can be employed as the base of a number system, for example, the base 10 (i.e., decimal) system, the base 8 (i.e., octal) system, the base 2 (i.e., binary) system, etc. The base 10, or decimal, system is the most common system used for human communication. Digital computers use the base 2, or binary, system. In a digital computer, a binary number consists of a number of binary bits. The number of binary bits in a binary number determines the precision with which the binary number represents a decimal number. The most common size binary number is a 32 bit number, which can represent approximately seven digits of a decimal number. Some digital computers have 64 bit binary numbers, which can represent 13 to 14 decimal digits. In many engineering and scientific calculations, 32 bit arithmetic is adequate. However, in many other applications, 64 bit arithmetic is required. In a few special situations, 128 bit arithmetic may be required. On 32 bit computers, 64 bit arithmetic, or even 128 bit arithmetic, can be accomplished using software enhancements. Such calculations are called **double precision** or **quad precision**, respectively. Such software enhanced precision can require as much as 10 times the execution time of a single precision calculation.

Consequently, some care must be exercised when deciding whether or not higher precision arithmetic is required. All of the examples in this book are evaluated using 64 bit arithmetic to ensure that roundoff is not significant.

Except for integers and some fractions, all binary representations of decimal numbers are approximations, owing to the finite word length of binary numbers. Thus, some loss of precision in the binary representation of a decimal number is unavoidable. When binary numbers are combined in arithmetic operations such as addition, multiplication, etc., the true result is typically a longer binary number which cannot be represented exactly with the number of available bits in the binary number capability of the digital computer. Thus, the results are rounded off in the last available binary bit. This rounding off gives rise to **roundoff error**, which can accumulate as the number of calculations increases.

0.7 SOFTWARE PACKAGES AND LIBRARIES

Numerous commercial software packages and libraries are available for implementing the numerical solution of engineering and scientific problems. Two of the more versatile software packages are Mathcad and Matlab. These software packages, as well as several other packages and several libraries, are listed below with a brief description of each one and references to sources for the software packages and libraries.

A. Software Packages

Excel Excel is a spreadsheet developed by Microsoft, Inc., as part of Microsoft Office. It enables calculations to be performed on rows and columns of numbers. The calculations to be performed are specified for each column. When any number on the spreadsheet is changed, all of the calculations are updated. Excel contains several built-in numerical algorithms. It also includes the Visual Basic programming language and some plotting capability. Although its basic function is not numerical analysis, Excel can be used productively for many types of numerical problems. Microsoft, Inc. www.microsoft.com/office/Excel.

Macsyma Macsyma is the world's first artificial intelligence based math engine providing easy to use, powerful math software for both symbolic and numerical computing. Macsyma, Inc., 20 Academy St., Arlington, MA 02476-6412. (781) 646-4550, webmaster@macsyma.com, www.macsyma.com.

Maple Maple 6 is a technologically advanced computational system with both algorithms and numeric solvers. Maple 6 includes an extensive set of NAG (Numerical Algorithms Group) solvers for computational linear algebra. Waterloo Maple, Inc., 57 Erb Street W., Waterloo, Ontario, Canada N2L 6C2. (800) 267-6583, (519) 747-2373, info@maplesoft.com, www.maplesoft.com.

Mathematica Mathematica 4 is a comprehensive software package which performs both symbolic and numeric computations. It includes a flexible and intuitive programming language and comprehensive plotting capabilities. Wolfram Research, Inc., 100 Trade Center Drive, Champaign IL 61820-7237. (800) 965-3726, (217) 398-0700, info@wolfram.com, www.wolfram.com.

Mathcad Mathcad 8 provides a free-form interface which permits the integration of real math notation, graphs, and text within a single interactive worksheet. It includes statistical and data analysis functions, powerful solvers, advanced matrix manipulation,

and the capability to create your own functions. Mathsoft, Inc., 101 Main Street, Cambridge, MA 02142-1521. (800) 628-4223, (617) 577-1017, info@mathsoft.com, www.mathcad.com.

Matlab Matlab is an integrated computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language. It provides core mathematics and advanced graphics tools for data analysis, visualization, and algorithm and application development, with more than 500 mathematical, statistical, and engineering functions. The Mathworks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2090. (508) 647-7000, info@mathworks.com, www.mathworks.com.

B. Libraries

GAMS GAMS (Guide to Available Mathematical Software) is a guide to over 9000 software modules contained in some 80 software packages at NIST (National Institute for Standards and Technology) and NETLIB. gams.nist.gov.

IMSL IMSL (International Mathematics and Statistical Library) is a comprehensive resource of more than 900 FORTRAN subroutines for use in general mathematics and statistical data analysis. Also available in C and Java. Visual Numerics, Inc., 1300 W. Sam Houston Parkway S., Suite 150, Houston TX 77042. (800) 364-8880, (713) 781-9260, info@houston.vni.com, www.vni.com.

LAPACK LAPACK is a library of FORTRAN 77 subroutines for solving linear algebra problems and eigenproblems. Individual subroutines can be obtained through NETLIB. The complete package can be obtained from NAG.

NAG NAG is a mathematical software library that contains over 1000 mathematical and statistical functions. Available in FORTRAN and C. NAG, Inc., 1400 Opus Place, Suite 200, Downers Grove, IL 60515-5702. (630) 971-2337, naginfo@nag.com, www.nag.com.

NETLIB NETLIB is a large collection of numerical libraries. netlib@research.att.com, netlib@ornl.gov, netlib@nec.no.

C. Numerical Recipes

Numerical Recipes is a book by William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. It contains over 300 subroutines for numerical algorithms. Versions of the subroutines are available in FORTRAN, C, Pascal, and Basic. The source codes are available on disk. Cambridge University Press, 40 West 20th Street, New York, NY 10011. www.cup.org.

0.8 THE TAYLOR SERIES AND THE TAYLOR POLYNOMIAL

A power series in powers of x is a series of the form

$$\sum_{n=0}^{\infty} a_n x^n = a_0 + a_1 x + a_2 x^2 + \dots \quad (0.1)$$

A power series in powers of $(x - x_0)$ is given by

$$\sum_{n=0}^{\infty} a_n (x - x_0)^n = a_0 + a_1 (x - x_0) + a_2 (x - x_0)^2 + \dots \quad (0.2)$$

Within its radius of convergence, r , any continuous function, $f(x)$, can be represented exactly by a power series. Thus,

$$f(x) = \sum_{n=0}^{\infty} a_n (x - x_0)^n \quad (0.3)$$

is continuous for $(x_0 - r) < x < (x_0 + r)$.

A. Taylor Series in One Independent Variable

If the coefficients, a_n , in Eq. (0.3) are given by the rule:

$$a_0 = f(x_0), a_1 = \frac{1}{1!} f'(x_0), a_2 = \frac{1}{2!} f''(x_0), \dots \quad (0.4)$$

then Eq. (0.3) becomes the **Taylor series** of $f(x)$ at $x = x_0$. Thus,

$$f(x) = f(x_0) + \frac{1}{1!} f'(x_0)(x - x_0) + \frac{1}{2!} f''(x_0)(x - x_0)^2 + \dots \quad (0.5)$$

Equation (0.5) can be written in the simpler appearing form

$$f(x) = f_0 + f'_0 \Delta x + \frac{1}{2} f''_0 \Delta x^2 + \dots + \frac{1}{n!} f_0^{(n)} \Delta x^n + \dots \quad (0.6)$$

where $f_0 = f(x_0)$, $f^{(n)} = df^{(n)}/dx^n$, and $\Delta x = (x - x_0)$. Equation (0.6) can be written in the compact form

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!} f_0^{(n)} (x - x_0)^n \quad (0.7)$$

When $x_0 = 0$, the Taylor series is known as the **Maclaurin series**. In that case, Eqs. (0.5) and (0.7) become

$$f(x) = f(0) + f'(0)x + \frac{1}{2} f''(0)x^2 + \dots \quad (0.8)$$

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(0) x^n \quad (0.9)$$

It is, of course, impractical to evaluate an infinite Taylor series term by term. The Taylor series can be written as the finite Taylor series, also known as the *Taylor formula* or *Taylor polynomial* with remainder, as follows:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(x_0)(x - x_0)^2 + \dots$$

$$+ \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n + R^{n+1} \quad (0.10)$$

where the term R^{n+1} is the **remainder term** given by

$$R^{n+1} = \frac{1}{(n+1)!} f^{(n+1)}(\xi) (x - x_0)^{n+1} \quad (0.11)$$

where ξ lies between x_0 and x . Equation (0.10) is quite useful in numerical analysis, where an approximation of $f(x)$ is obtained by truncating the remainder term.

B. Taylor Series in Two Independent Variables

Power series can also be written for functions of more than one independent variable. For a function of two independent variables, $f(x, y)$, the Taylor series of $f(x, y)$ at (x_0, y_0) is given by

$$\boxed{f(x, y) = f_0 + \frac{\partial f}{\partial x} \Big|_0 (x - x_0) + \frac{\partial f}{\partial y} \Big|_0 (y - y_0) + \frac{1}{2!} \left(\frac{\partial^2 f}{\partial x^2} \Big|_0 (x - x_0)^2 + 2 \frac{\partial^2 f}{\partial x \partial y} \Big|_0 (x - x_0)(y - y_0) + \frac{\partial^2 f}{\partial y^2} \Big|_0 (y - y_0)^2 \right) + \dots} \quad (0.12)$$

Equation (0.12) can be written in the general form

$$f(x, y) = \sum_{n=0}^{\infty} \frac{1}{n!} \left((x - x_0) \frac{\partial}{\partial x} + (y - y_0) \frac{\partial}{\partial y} \right)^n f(x, y)|_0 \quad (0.13)$$

where the term $(\dots)^n$ is expanded by the binomial expansion and the resulting expansion operates on the function $f(x, y)$ and is evaluated at (x_0, y_0) .

The Taylor formula with remainder for a function of two independent variables is obtained by evaluating the derivatives in the $(n+1)$ st term at the point (ξ, η) , where (ξ, η) lies in the region between points (x_0, y_0) and (x, y) .

Basic Tools of Numerical Analysis

-
- I.1. Systems of Linear Algebraic Equations
 - I.2. Eigenproblems
 - I.3. Roots of Nonlinear Equations
 - I.4. Polynomial Approximation and Interpolation
 - I.5. Numerical Differentiation and Difference Formulas
 - I.6. Numerical Integration
 - I.7. Summary

Many different types of algebraic processes are required in engineering and science. These processes include the solution of systems of linear algebraic equations, the solution of eigenproblems, finding the roots of nonlinear equations, polynomial approximation and interpolation, numerical differentiation and difference formulas, and numerical integration. These topics are not only important in their own right, they lay the foundation for the solution of ordinary and partial differential equations, which are discussed in Parts II and III, respectively. Figure I.1 illustrates the types of problems considered in Part I.

The objective of Part I is to introduce and discuss the general features of each of these algebraic processes, which are the *basic tools of numerical analysis*.

I.1 SYSTEMS OF LINEAR ALGEBRAIC EQUATIONS

Systems of equations arise in all branches of engineering and science. These equations may be algebraic, transcendental (i.e., involving trigonometric, logarithmic, exponential, etc., functions), ordinary differential equations, or partial differential equations. The equations may be linear or nonlinear. Chapter 1 is devoted to the solution of *systems of linear algebraic equations* of the following form:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \quad (\text{I.1a})$$

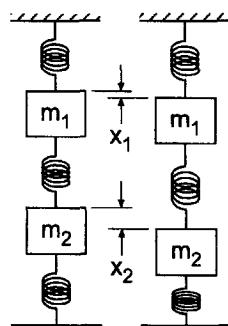
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \quad (\text{I.1b})$$

.....

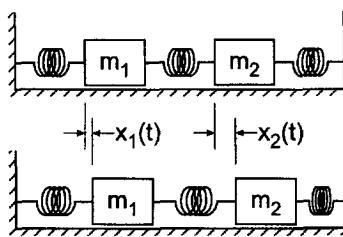
$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \quad (\text{I.1n})$$

where x_j ($j = 1, 2, \dots, n$) denotes the unknown variables, $a_{i,j}$ ($i, j = 1, 2, \dots, n$) denotes the coefficients of the unknown variables, and b_i ($i = 1, 2, \dots, n$) denotes the nonhomogeneous terms. For the coefficients $a_{i,j}$, the first subscript i corresponds to equation i , and the second subscript j corresponds to variable x_j . The number of equations can range from two to hundreds, thousands, and even millions.

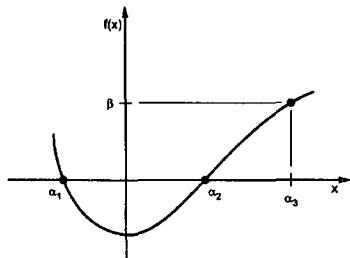
Systems of linear algebraic equations arise in many different problems, for example, (a) network problems (e.g., electrical networks), (b) fitting approximating functions (see Chapter 4), and (c) systems of finite difference equations that arise in the numerical solution of differential equations (see Chapters 7 to 12). The list is endless. Figure I.1a illustrates a static spring-mass system, whose static equilibrium configuration is governed by a system of linear algebraic equations. That system of equations is used throughout Chapter 1 as an example problem.



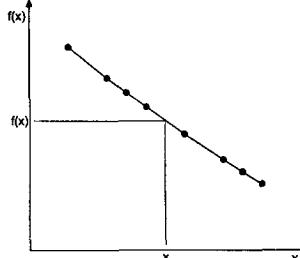
(a) Static spring-mass system.



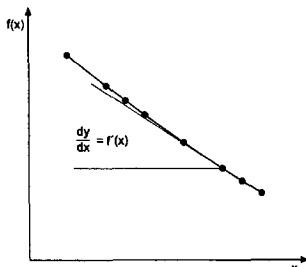
(b) Dynamic spring-mass system.



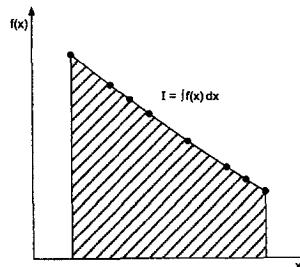
(c) Roots of nonlinear equations. (d) Polynomial approximation and interpolation.



(c) Roots of nonlinear equations. (d) Polynomial approximation and interpolation.



(e) Numerical differentiation.



(f) Numerical integration.

Figure I.1 Basic tools of numerical analysis. (a) Static spring-mass system. (b) Dynamic spring-mass system. (c) Roots of nonlinear equations. (d) Polynomial approximation and interpolation. (e) Numerical differentiation. (f) Numerical integration.

Systems of linear algebraic equations can be expressed very conveniently in terms of matrix notation. Solution methods can be developed very compactly in terms of matrix notation. Consequently, the elementary properties of matrices and determinants are reviewed at the beginning of Chapter 1.

Two fundamentally different approaches can be used to solve systems of linear algebraic equations:

1. Direct methods
2. Iterative methods

Direct methods are systematic procedures based on algebraic elimination. Several direct elimination methods, for example, Gauss elimination, are presented in Chapter 1. Iterative methods obtain the solution asymptotically by an iterative procedure in which a trial solution is assumed, the trial solution is substituted into the system of equations to determine the mismatch, or error, and an improved solution is obtained from the mismatch data. Several iterative methods, for example, successive-over-relaxation (SOR), are presented in Chapter 1.

The notation, concepts, and procedures presented in Chapter 1 are used throughout the remainder of the book. A solid understanding of systems of linear algebraic equations is essential in numerical analysis.

I.2 EIGENPROBLEMS

Eigenproblems arise in the special case where a system of algebraic equations is homogeneous; that is, the nonhomogeneous terms, b_i in Eq. (I.1), are all zero, and the coefficients contain an unspecified parameter, say λ . In general, when $b_i = 0$, the only solution to Eq. (I.1) is the trivial solution, $x_1 = x_2 = \dots = x_n = 0$. However, when the coefficients $a_{i,j}$ contain an unspecified parameter, say λ , the value of that parameter can be chosen so that the system of equations is redundant, and an infinite number of solutions exist. The unspecified parameter λ is an *eigenvalue* of the system of equations. For example,

$$(a_{11} - \lambda)x_1 + a_{12}x_2 = 0 \quad (\text{I.2a})$$

$$a_{21}x_1 + (a_{22} - \lambda)x_2 = 0 \quad (\text{I.2b})$$

is a linear eigenproblem. The value (or values) of λ that make Eqs. (I.2a) and (I.2b) identical are the eigenvalues of Eqs. (I.2). In that case, the two equations are redundant, so the only unique solution is $x_1 = x_2 = 0$. However, an infinite number of solutions can be obtained by specifying either x_1 or x_2 , then calculating the other from either of the two redundant equations. The set of values of x_1 and x_2 corresponding to a particular value of λ is an *eigenvector* of Eq. (I.2). Chapter 2 is devoted to the solution of eigenproblems.

Eigenproblems arise in the analysis of many physical systems. They arise in the analysis of the dynamic behavior of mechanical, electrical, fluid, thermal, and structural systems. They also arise in the analysis of control systems. Figure I.1b illustrates a dynamic spring-mass system, whose dynamic equilibrium configuration is governed by a system of homogeneous linear algebraic equations. That system of equations is used throughout Chapter 2 as an example problem. When the static equilibrium configuration of the system is disturbed and then allowed to vibrate freely, the system of masses will oscillate at special frequencies, which depend on the values of the masses and the spring

constants. These special frequencies are the eigenvalues of the system. The relative values of x_1 , x_2 , etc. corresponding to each eigenvalue λ are the eigenvectors of the system.

The objectives of Chapter 2 are to introduce the general features of eigenproblems and to present several methods for solving eigenproblems. Eigenproblems are special problems of interest only in themselves. Consequently, an understanding of eigenproblems is not essential to the other concepts presented in this book.

I.3 ROOTS OF NONLINEAR EQUATIONS

Nonlinear equations arise in many physical problems. Finding their roots, or zeros, is a common problem. The problem can be stated as follows:

Given the continuous nonlinear function $f(x)$, find the value of $x = \alpha$ such that

$$f(\alpha) = 0$$

where α is the *root*, or zero, of the nonlinear equation. Figure I.1c illustrates the problem graphically. The function $f(x)$ may be an algebraic function, a transcendental function, the solution of a differential equation, or any nonlinear relationship between an input x and a response $f(x)$. Chapter 3 is devoted to the solution of nonlinear equations.

Nonlinear equations are solved by iterative methods. A trial solution is assumed, the trial solution is substituted into the nonlinear equation to determine the error, or mismatch, and the mismatch is used in some systematic manner to generate an improved estimate of the solution. Several methods for finding the roots of nonlinear equations are presented in Chapter 3. The workhorse methods of choice for solving nonlinear equations are Newton's method and the secant method. A detailed discussion of finding the roots of polynomials is presented. A brief introduction to the problems of solving systems of nonlinear equations is also presented.

Nonlinear equations occur throughout engineering and science. Nonlinear equations also arise in other areas of numerical analysis. For example, the shooting method for solving boundary-value ordinary differential equations, presented in Section 8.3, requires the solution of a nonlinear equation. Implicit methods for solving nonlinear differential equations yield nonlinear difference equations. The solution of such problems is discussed in Sections 7.11, 8.7, 9.11, 10.9, and 11.8. Consequently, a thorough understanding of methods for solving nonlinear equations is an essential requirement for the numerical analyst.

I.4 POLYNOMIAL APPROXIMATION AND INTERPOLATION

In many problems in engineering and science, the data under consideration are known only at discrete points, not as a continuous function. For example, as illustrated in Figure I.1d, the continuous function $f(x)$ may be known only at n discrete values of x :

$$y_i = y(x_i) \quad (i = 1, 2, \dots, n) \tag{I.3}$$

Values of the function at points other than the known discrete points may be needed (i.e., *interpolation*). The derivative of the function at some point may be needed (i.e., *differentiation*). The integral of the function over some range may be required (i.e., *integration*). These processes, for discrete data, are performed by fitting an approximating function to the set of discrete data and performing the desired processes on the approximating function. Many types of approximating functions can be used.

Because of their simplicity, ease of manipulation, and ease of evaluation, polynomials are an excellent choice for an approximating function. The general n th-degree polynomial is specified by

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \quad (I.4)$$

Polynomials can be fit to a set of discrete data in two ways:

1. Exact fit
2. Approximate fit

An exact fit passes exactly through all the discrete data points. Direct fit polynomials, divided-difference polynomials, and Lagrange polynomials are presented in Chapter 4 for fitting nonequally spaced data or equally spaced data. Newton difference polynomials are presented for fitting equally spaced data. The least squares procedure is presented for determining approximate polynomial fits.

Figure I.1d illustrates the problem of interpolating within a set of discrete data. Procedures for interpolating within a set of discrete data are presented in Chapter 4.

Polynomial approximation is essential for interpolation, differentiation, and integration of sets of discrete data. A good understanding of polynomial approximation is a necessary requirement for the numerical analyst.

I.5 NUMERICAL DIFFERENTIATION AND DIFFERENCE FORMULAS

The evaluation of derivatives, a process known as *differentiation*, is required in many problems in engineering and science. Differentiation of the function $f(x)$ is denoted by

$$\frac{d}{dx}(f(x)) = f'(x) \quad (I.5)$$

The function $f(x)$ may be a known function or a set of discrete data. In general, known functions can be differentiated exactly. Differentiation of discrete data requires an approximate numerical procedure. Numerical differentiation formulas can be developed by fitting approximating functions (e.g., polynomials) to a set of discrete data and differentiating the approximating function. For polynomial approximating functions, this yields

$$\frac{d}{dx}(f(x)) = f'(x) \cong \frac{d}{dx}(P_n(x)) = P'_n(x) \quad (I.6)$$

Figure I.1e illustrates the problem of numerical differentiation of a set of discrete data. Numerical differentiation procedures are developed in Chapter 5.

The approximating polynomial may be fit exactly to a set of discrete data by the methods presented in Chapter 4, or fit approximately by the least squares procedure described in Chapter 4. Several numerical differentiation formulas based on differentiation of polynomials are presented in Chapter 5.

Numerical differentiation formulas also can be developed using Taylor series. This approach is quite useful for developing difference formulas for approximating exact derivatives in the numerical solution of differential equations. Section 5.5 presents a table of difference formulas for use in the solution of differential equations.

Numerical differentiation of discrete data is not required very often. However, the numerical solution of differential equations, which is the subject of Parts II and III, is one

of the most important areas of numerical analysis. The use of difference formulas is essential in that application.

I.6 NUMERICAL INTEGRATION

The evaluation of integrals, a process known as *integration*, or quadrature, is required in many problems in engineering and science. Integration of the function $f(x)$ is denoted by

$$I = \int_a^b f(x) dx \quad (I.7)$$

The function $f(x)$ may be a known function or a set of discrete data. Some known functions have an exact integral. Many known functions, however, do not have an exact integral, and an approximate numerical procedure is required to evaluate Eq. (I.7). When a known function is to be integrated numerically, it must first be discretized. Integration of discrete data always requires an approximate numerical procedure. Numerical integration (quadrature) formulas can be developed by fitting approximating functions (e.g., polynomials) to a set of discrete data and integrating the approximating function. For polynomial approximating functions, this gives

$$I = \int_a^b f(x) dx \cong \int_a^b P_n(x) dx \quad (I.8)$$

Figure I.1f illustrates the problem of numerical integration of a set of discrete data. Numerical integration procedures are developed in Chapter 6.

The approximating function can be fit exactly to a set of discrete data by direct fit methods, or fit approximately by the least squares method. For unequally spaced data, direct fit polynomials can be used. For equally spaced data, the Newton forward-difference polynomials of different degrees can be integrated to yield the Newton-Cotes quadrature formulas. The most prominent of these are the trapezoid rule and Simpson's 1/3 rule. Romberg integration, which is a higher-order extrapolation of the trapezoid rule, is introduced. Adaptive integration, in which the range of integration is subdivided automatically until a specified accuracy is obtained, is presented. Gaussian quadrature, which achieves higher-order accuracy for integrating known functions by specifying the locations of the discrete points, is presented. The evaluation of multiple integrals is discussed.

Numerical integration of both known functions and discrete data is a common problem. The concepts involved in numerical integration lead directly to numerical methods for solving differential equations.

I.7 SUMMARY

Part I of this book is devoted to the *basic tools of numerical analysis*. These topics are important in their own right. In addition, they provide the foundation for the solution of ordinary and partial differential equations, which are discussed in Parts II and III, respectively. The material presented in Part I comprises the basic language of numerical analysis. Familiarity and mastery of this material is essential for the understanding and use of more advanced numerical methods.

1

Systems of Linear Algebraic Equations

- 1.1. Introduction
- 1.2. Properties of Matrices and Determinants
- 1.3. Direct Elimination Methods
- 1.4. LU Factorization
- 1.5. Tridiagonal Systems of Equations
- 1.6. Pitfalls of Elimination Methods
- 1.7. Iterative Methods
- 1.8. Programs
- 1.9. Summary
Problems

Examples

- 1.1. Matrix addition
- 1.2. Matrix multiplication
- 1.3. Evaluation of a 3×3 determinant by the diagonal method
- 1.4. Evaluation of a 3×3 determinant by the cofactor method
- 1.5. Cramer's rule
- 1.6. Elimination
- 1.7. Simple elimination
- 1.8. Simple elimination for multiple \mathbf{b} vectors
- 1.9. Elimination with pivoting to avoid zero pivot elements
- 1.10. Elimination with scaled pivoting to reduce round-off errors
- 1.11. Gauss-Jordan elimination
- 1.12. Matrix inverse by Gauss-Jordan elimination
- 1.13. The matrix inverse method
- 1.14. Evaluation of a 3×3 determinant by the elimination method
- 1.15. The Doolittle LU method
- 1.16. Matrix inverse by the Doolittle LU method
- 1.17. The Thomas algorithm
- 1.18. Effects of round-off errors
- 1.19. System condition
- 1.20. Norms and condition numbers
- 1.21. The Jacobi iteration method
- 1.22. The Gauss-Seidel iteration method
- 1.23. The SOR method

1.1 INTRODUCTION

The static mechanical spring-mass system illustrated in Figure 1.1 consists of three masses m_1 to m_3 , having weights W_1 to W_3 , interconnected by five linear springs K_1 to K_5 . In the configuration illustrated on the left, the three masses are supported by forces F_1 to F_3 equal to weights W_1 to W_3 , respectively, so that the five springs are in a stable static equilibrium configuration. When the supporting forces F_1 to F_3 are removed, the masses move downward and reach a new static equilibrium configuration, denoted by x_1 , x_2 , and x_3 , where x_1 , x_2 , and x_3 are measured from the original locations of the corresponding masses. Free-body diagrams of the three masses are presented at the bottom of Figure 1.1. Performing a static force balance on the three masses yields the following system of three linear algebraic equations:

$$(K_1 + K_2 + K_3)x_1 - K_2x_2 - K_3x_3 = W_1 \quad (1.1a)$$

$$-K_2x_1 + (K_2 + K_4)x_2 - K_4x_3 = W_2 \quad (1.1b)$$

$$-K_3x_1 - K_4x_2 + (K_3 + K_4 + K_5)x_3 = W_3 \quad (1.1c)$$

When values of K_1 to K_5 and W_1 to W_3 are specified, the equilibrium displacements x_1 to x_3 can be determined by solving Eq. (1.1).

The static mechanical spring-mass system illustrated in Figure 1.1 is used as the example problem in this chapter to illustrate methods for solving systems of linear

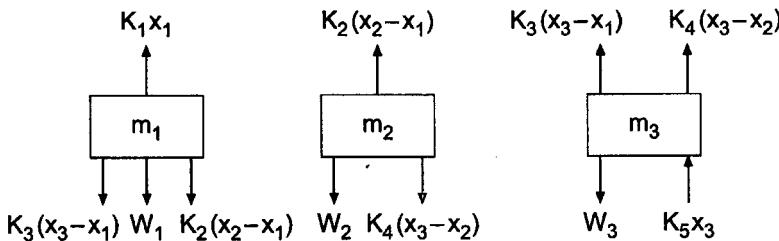
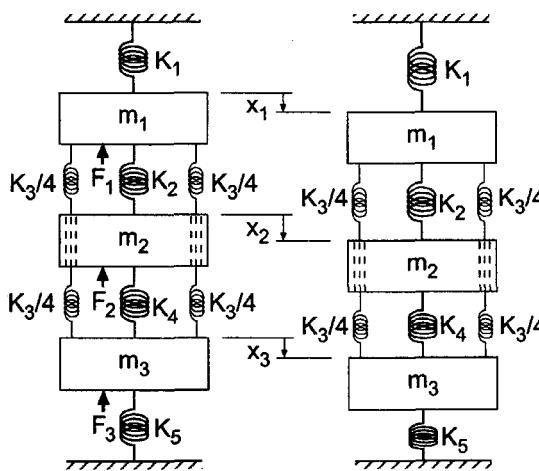


Figure 1.1 Static mechanical spring-mass system.

algebraic equations. For that purpose, let $K_1 = 40 \text{ N/cm}$, $K_2 = K_3 = K_4 = 20 \text{ N/cm}$, and $K_5 = 90 \text{ N/cm}$. Let $W_1 = W_2 = W_3 = 20 \text{ N}$. For these values, Eq. (1.1) becomes:

$$80x_1 - 20x_2 - 20x_3 = 20 \quad (1.2a)$$

$$-20x_1 + 40x_2 - 20x_3 = 20 \quad (1.2b)$$

$$-20x_1 - 20x_2 + 130x_3 = 20 \quad (1.2c)$$

The solution to Eq. (1.2) is $x_1 = 0.6 \text{ cm}$, $x_2 = 1.0 \text{ cm}$, and $x_3 = 0.4 \text{ cm}$, which can be verified by direct substitution.

Systems of equations arise in all branches of engineering and science. These equations may be algebraic, transcendental (i.e., involving trigonometric, logarithmic, exponential, etc. functions), ordinary differential equations, or partial differential equations. The equations may be linear or nonlinear. Chapter 1 is devoted to the solution of systems of linear algebraic equations of the following form:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \quad (1.3a)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \quad (1.3b)$$

.....

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \quad (1.3n)$$

where x_j ($j = 1, 2, \dots, n$) denotes the unknown variables, $a_{i,j}$ ($i, j = 1, 2, \dots, n$) denotes the constant coefficients of the unknown variables, and b_i ($i = 1, 2, \dots, n$) denotes the nonhomogeneous terms. For the coefficients $a_{i,j}$, the first subscript, i , denotes equation i , and the second subscript, j , denotes variable x_j . The number of equations can range from two to hundreds, thousands, and even millions.

In the most general case, the number of variables is not required to be the same as the number of equations. However, in most practical problems, they are the same. That is the case considered in this chapter. Even when the number of variables is the same as the number of equations, several solution possibilities exist, as illustrated in Figure 1.2 for the following system of two linear algebraic equations:

$$a_{11}x_1 + a_{12}x_2 = b_1 \quad (1.4a)$$

$$a_{21}x_1 + a_{22}x_2 = b_2 \quad (1.4b)$$

The four solution possibilities are:

1. A unique solution (a consistent set of equations), as illustrated in Figure 1.2a
2. No solution (an inconsistent set of equations), as illustrated in Figure 1.2b
3. An infinite number of solutions (a redundant set of equations), as illustrated in Figure 1.2c
4. The trivial solution, $x_j = 0$ ($j = 1, 2, \dots, n$), for a homogeneous set of equations, as illustrated in Figure 1.2d

Chapter 1 is concerned with the first case where a unique solution exists.

Systems of linear algebraic equations arise in many different types of problems, for example:

1. Network problems (e.g., electrical networks)
2. Fitting approximating functions (see Chapter 4)

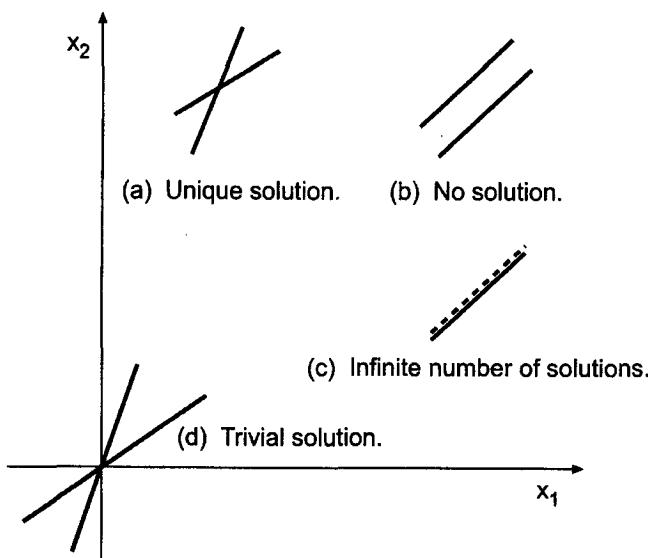


Figure 1.2 Solution of a system of two linear algebraic equations.

3. Systems of finite difference equations that arise in the numerical solution of differential equations (see Parts II and III)

The list is endless.

There are two fundamentally different approaches for solving systems of linear algebraic equations:

1. Direct elimination methods
2. Iterative methods

Direct elimination methods are systematic procedures based on algebraic elimination, which obtain the solution in a fixed number of operations. Examples of direct elimination methods are *Gauss elimination*, *Gauss-Jordan elimination*, the *matrix inverse method*, and *Doolittle LU factorization*. Iterative methods, on the other hand, obtain the solution asymptotically by an iterative procedure. A trial solution is assumed, the trial solution is substituted into the system of equations to determine the mismatch, or error, in the trial solution, and an improved solution is obtained from the mismatch data. Examples of iterative methods are *Jacobi iteration*, *Gauss-Seidel iteration*, and *successive-over-relaxation (SOR)*.

Although no absolutely rigid rules apply, direct elimination methods are generally used when one or more of the following conditions holds: (a) The number of equations is small (100 or less), (b) most of the coefficients in the equations are nonzero, (c) the system of equations is not diagonally dominant [see Eq. (1.15)], or (d) the system of equations is ill conditioned (see Section 1.6.2). Iterative methods are used when the number of equations is large and most of the coefficients are zero (i.e., a sparse matrix). Iterative methods generally diverge unless the system of equations is diagonally dominant [see Eq. (1.15)].

The organization of Chapter 1 is illustrated in Figure 1.3. Following the introductory material discussed in this section, the properties of matrices and determinants are reviewed. The presentation then splits into a discussion of direct elimination methods

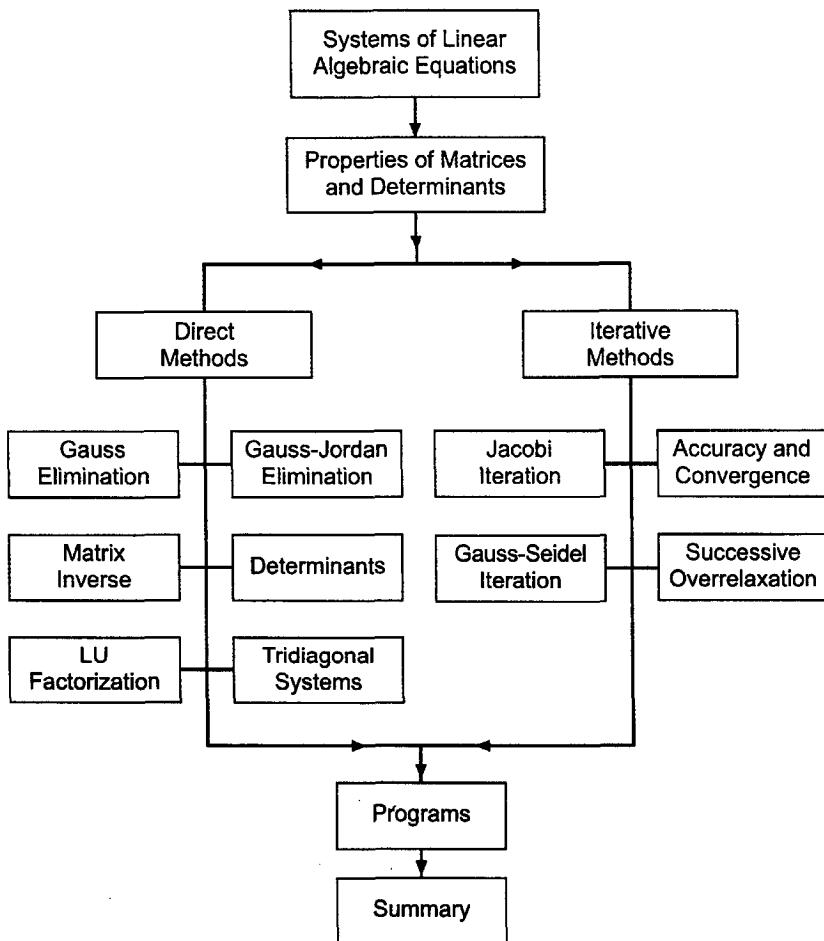


Figure 1.3 Organization of Chapter 1.

followed by a discussion of iterative methods. Several methods, both direct elimination and iterative, for solving systems of linear algebraic equations are presented in this chapter. Procedures for special problems, such as tridiagonal systems of equations, are presented. All these procedures are illustrated by examples. Although the methods apply to large systems of equations, they are illustrated by applying them to the small system of only three equations given by Eq. (1.2). After the presentation of the methods, three computer programs are presented for implementing the Gauss elimination method, the Thomas algorithm, and successive-over-relaxation (SOR). The chapter closes with a Summary, which discusses some philosophy to help you choose the right method for every problem and lists the things you should be able to do after studying Chapter 1.

1.2 PROPERTIES OF MATRICES AND DETERMINANTS

Systems of linear algebraic equations can be expressed very conveniently in terms of matrix notation. Solution methods for systems of linear algebraic equations can be

developed very compactly using matrix algebra. Consequently, the elementary properties of matrices and determinants are presented in this section.

1.2.1. Matrix Definitions

A *matrix* is a rectangular array of elements (either numbers or symbols), which are arranged in orderly rows and columns. Each element of the matrix is distinct and separate. The location of an element in the matrix is important. Elements of a matrix are generally identified by a double subscripted lowercase letter, for example, $a_{i,j}$, where the first subscript i identifies the row of the matrix and the second subscript j identifies the column of the matrix. The size of a matrix is specified by the number of rows times the number of columns. A matrix with n rows and m columns is said to be an n by m , or $n \times m$, matrix. Matrices are generally represented by either a boldface capital letter, for example, \mathbf{A} , the general element enclosed in brackets, for example, $[a_{i,j}]$, or the full array of elements, as illustrated in Eq. (1.5):

$$\mathbf{A} = [a_{i,j}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2m} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \cdots & \cdots & a_{nm} \end{bmatrix} \quad (i = 1, 2, \dots, n; j = 1, 2, \dots, m) \quad (1.5)$$

Comparing Eqs. (1.3) and (1.5) shows that the coefficients of a system of linear algebraic equations form the elements of an $n \times n$ matrix.

Equation (1.5) illustrates a convention used throughout this book for simplicity of appearance. When the general element $a_{i,j}$ is considered, the subscripts i and j are separated by a comma. When a specific element is specified, for example, a_{31} , the subscripts 3 and 1, which denote the element in row 3 and column 1, will not be separated by a comma, unless i or j is greater than 9. For example, a_{37} denotes the element in row 3 and column 7, whereas $a_{13,17}$ denotes the element in row 13 and column 17.

Vectors are a special type of matrix which has only one column or one row. Vectors are represented by either a boldface lowercase letter, for example, \mathbf{x} or \mathbf{y} , the general element enclosed in brackets, for example, $[x_i]$ or $[y_i]$, or the full column or row of elements. A *column vector* is an $n \times 1$ matrix. Thus,

$$\mathbf{x} = [x_i] = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad (i = 1, 2, \dots, n) \quad (1.6a)$$

A *row vector* is a $1 \times n$ matrix. For example,

$$\mathbf{y} = [y_j] = [y_1 \ y_2 \ \cdots \ y_n] \quad (j = 1, 2, \dots, n) \quad (1.6b)$$

Unit vectors, \mathbf{i} , are special vectors which have a magnitude of unity. Thus,

$$\|\mathbf{i}\| = (i_1^2 + i_2^2 + \cdots + i_n^2)^{1/2} = 1 \quad (1.7)$$

where the notation $\|\mathbf{i}\|$ denotes the length of vector \mathbf{i} . Orthogonal systems of unit vectors, in which all of the elements of each unit vector except one are zero, are used to define coordinate systems.

There are several special matrices of interest. A *square matrix* \mathbf{S} is a matrix which has the same number of rows and columns, that is, $m = n$. For example,

$$\mathbf{S} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad (1.8)$$

is a square $n \times n$ matrix. Our interest will be devoted entirely to square matrices. The left-to-right downward-sloping line of elements from a_{11} to a_{nn} is called the *major diagonal* of the matrix. A *diagonal matrix* \mathbf{D} is a square matrix with all elements equal to zero except the elements on the major diagonal. For example,

$$\mathbf{D} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \quad . \quad (1.9)$$

is a 4×4 diagonal matrix. The *identity matrix* \mathbf{I} is a diagonal matrix with unity diagonal elements. The identity matrix is the matrix equivalent of the scalar number unity. The matrix

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1.10)$$

is the 4×4 identity matrix.

A *triangular matrix* is a square matrix in which all of the elements on one side of the major diagonal are zero. The remaining elements may be zero or nonzero. An *upper triangular matrix* \mathbf{U} has all zero elements below the major diagonal. The matrix

$$\mathbf{U} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix} \quad (1.11)$$

is a 4×4 upper triangular matrix. A *lower triangular matrix* \mathbf{L} has all zero elements above the major diagonal. The matrix

$$\mathbf{L} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad (1.12)$$

is a 4×4 lower triangular matrix.

A *tridiagonal matrix* \mathbf{T} is a square matrix in which all of the elements not on the major diagonal and the two diagonals surrounding the major diagonal are zero. The elements on these three diagonals may or may not be zero. The matrix

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix} \quad (1.13)$$

is a 5×5 tridiagonal matrix.

A *banded matrix* \mathbf{B} has all zero elements except along particular diagonals. For example,

$$\mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & 0 & a_{14} & 0 \\ a_{21} & a_{22} & a_{23} & 0 & a_{25} \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ a_{41} & 0 & a_{43} & a_{44} & a_{45} \\ 0 & a_{52} & 0 & a_{54} & a_{55} \end{bmatrix} \quad (1.14)$$

is a 5×5 banded matrix.

The *transpose* of an $n \times m$ matrix \mathbf{A} is the $m \times n$ matrix, \mathbf{A}^T , which has elements $a_{i,j}^T = a_{j,i}$. The transpose of a column vector, is a row vector and vice versa. *Symmetric* square matrices have identical corresponding elements on either side of the major diagonal. That is, $a_{i,j} = a_{j,i}$. In that case, $\mathbf{A} = \mathbf{A}^T$.

A *sparse matrix* is one in which most of the elements are zero. Most large matrices arising in the solution of ordinary and partial differential equations are sparse matrices.

A matrix is *diagonally dominant* if the absolute value of each element on the major diagonal is equal to, or larger than, the sum of the absolute values of all the other elements in that row, with the diagonal element being larger than the corresponding sum of the other elements for at least one row. Thus, diagonal dominance is defined as

$$|a_{i,i}| \geq \sum_{j=1, j \neq i}^n |a_{i,j}| \quad (i = 1, \dots, n) \quad (1.15)$$

with $>$ true for at least one row.

1.2.2. Matrix Algebra

Matrix algebra consists of *matrix addition*, *matrix subtraction*, and *matrix multiplication*. Matrix division is not defined. An analogous operation is accomplished using the matrix inverse.

Matrix addition and subtraction consist of adding or subtracting the corresponding elements of two matrices of equal size. Let \mathbf{A} and \mathbf{B} be two matrices of equal size. Then,

$$\mathbf{A} + \mathbf{B} = [a_{i,j}] + [b_{i,j}] = [a_{i,j} + b_{i,j}] = [c_{i,j}] = \mathbf{C} \quad (1.16a)$$

$$\mathbf{A} - \mathbf{B} = [a_{i,j}] - [b_{i,j}] = [a_{i,j} - b_{i,j}] = [c_{i,j}] = \mathbf{C} \quad (1.16b)$$

Unequal size matrices cannot be added or subtracted. Matrices of the same size are *associative* on addition. Thus,

$$\mathbf{A} + (\mathbf{B} + \mathbf{C}) = (\mathbf{A} + \mathbf{B}) + \mathbf{C} \quad (1.17)$$

Matrices of the same size are *commutative* on addition. Thus,

$$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A} \quad (1.18)$$

Example 1.1. Matrix addition.

Add the two 3×3 matrices \mathbf{A} and \mathbf{B} to obtain the 3×3 matrix \mathbf{C} , where

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 1 & 4 & 3 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 3 & 2 & 1 \\ -4 & 1 & 2 \\ 2 & 3 & -1 \end{bmatrix} \quad (1.19)$$

From Eq. (1.16a),

$$c_{ij} = a_{ij} + b_{ij} \quad (1.20)$$

Thus, $c_{11} = a_{11} + b_{11} = 1 + 3 = 4$, $c_{12} = a_{12} + b_{12} = 2 + 2 = 4$, etc. The result is

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} (1+3) & (2+2) & (3+1) \\ (2-4) & (1+1) & (4+2) \\ (1+2) & (4+3) & (3-1) \end{bmatrix} = \begin{bmatrix} 4 & 4 & 4 \\ -2 & 2 & 6 \\ 3 & 7 & 2 \end{bmatrix} = \mathbf{C} \quad (1.21)$$

Matrix multiplication consists of row-element to column-element multiplication and summation of the resulting products. Multiplication of the two matrices \mathbf{A} and \mathbf{B} is defined only when the number of columns of matrix \mathbf{A} is the same as the number of rows of matrix \mathbf{B} . Matrices that satisfy this condition are called *conformable* in the order \mathbf{AB} . Thus, if the size of matrix \mathbf{A} is $n \times m$ and the size of matrix \mathbf{B} is $m \times r$, then

$$\mathbf{AB} = [a_{ij}][b_{ij}] = [c_{ij}] = \mathbf{C} \quad c_{ij} = \sum_{k=1}^m a_{ik}b_{kj} \quad (i = 1, 2, \dots, n, j = 1, 2, \dots, r) \quad (1.22)$$

The size of matrix \mathbf{C} is $n \times r$. Matrices that are not conformable cannot be multiplied.

It is easy to make errors when performing matrix multiplication by hand. It is helpful to trace across the rows of \mathbf{A} with the left index finger while tracing down the columns of \mathbf{B} with the right index finger, multiplying the corresponding elements, and summing the products. Matrix algebra is much better suited to computers than to humans.

Multiplication of the matrix \mathbf{A} by the scalar α consists of multiplying each element of \mathbf{A} by α . Thus,

$$\alpha\mathbf{A} = \alpha[a_{ij}] = [\alpha a_{ij}] = [b_{ij}] = \mathbf{B} \quad (1.23)$$

Example 1.2. Matrix multiplication.

Multiply the 3×3 matrix \mathbf{A} and the 3×2 matrix \mathbf{B} to obtain the 3×2 matrix \mathbf{C} , where

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 1 & 4 & 3 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \\ 2 & 1 \end{bmatrix} \quad (1.24)$$

From Eq. (1.22),

$$c_{i,j} = \sum_{k=1}^3 a_{i,k} b_{k,j} \quad (i = 1, 2, 3, j = 1, 2) \quad (1.25)$$

Evaluating Eq. (1.25) yields

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} = (1)(2) + (2)(1) + (3)(2) = 10 \quad (1.26a)$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} = (1)(1) + (2)(2) + (3)(1) = 8 \quad (1.26b)$$

.....

$$c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} = (1)(1) + (4)(2) + (3)(1) = 12 \quad (1.26c)$$

Thus,

$$\mathbf{C} = [c_{i,j}] = \begin{bmatrix} 10 & 8 \\ 13 & 8 \\ 12 & 12 \end{bmatrix} \quad (1.27)$$

Multiply the 3×2 matrix \mathbf{C} by the scalar $\alpha = 2$ to obtain the 3×2 matrix \mathbf{D} . From Eq. (1.23), $d_{11} = \alpha c_{11} = (2)(10) = 20$, $d_{12} = \alpha c_{12} = (2)(8) = 16$, etc. The result is

$$\mathbf{D} = \alpha\mathbf{C} = 2\mathbf{C} = \begin{bmatrix} (2)(10) & (2)(8) \\ (2)(13) & (2)(8) \\ (2)(12) & (2)(12) \end{bmatrix} = \begin{bmatrix} 20 & 16 \\ 26 & 16 \\ 24 & 24 \end{bmatrix} \quad (1.28)$$

Matrices that are suitably conformable are *associative* on multiplication. Thus,

$$\mathbf{A}(\mathbf{B}\mathbf{C}) = (\mathbf{AB})\mathbf{C} \quad (1.29)$$

Square matrices are *conformable* in either order. Thus, if \mathbf{A} and \mathbf{B} are $n \times n$ matrices,

$$\mathbf{AB} = \mathbf{C} \quad \text{and} \quad \mathbf{BA} = \mathbf{D} \quad (1.30)$$

where \mathbf{C} and \mathbf{D} are $n \times n$ matrices. However square matrices in general are not *commutative* on multiplication. That is, in general,

$$\mathbf{AB} \neq \mathbf{BA} \quad (1.31)$$

Matrices \mathbf{A} , \mathbf{B} , and \mathbf{C} are *distributive* if \mathbf{B} and \mathbf{C} are the same size and \mathbf{A} is *conformable* to \mathbf{B} and \mathbf{C} . Thus,

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC} \quad (1.32)$$

Consider the two square matrices \mathbf{A} and \mathbf{B} . Multiplying yields

$$\mathbf{AB} = \mathbf{C} \quad (1.33)$$

It might appear logical that the inverse operation of multiplication, that is, division, would give

$$\mathbf{A} = \mathbf{C}/\mathbf{B} \quad (1.34)$$

Unfortunately, matrix division is not defined. However, for square matrices, an analogous concept is provided by the matrix inverse.

Consider the two square matrices \mathbf{A} and \mathbf{B} . If $\mathbf{AB} = \mathbf{I}$, then \mathbf{B} is the inverse of \mathbf{A} , which is denoted as \mathbf{A}^{-1} . Matrix inverses *commute* on multiplication. Thus,

$$\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I} \quad (1.35)$$

The operation desired by Eq. (1.34) can be accomplished using the matrix inverse. Thus, the inverse of the matrix multiplication specified by Eq. (1.33) is accomplished by matrix multiplication using the inverse matrix. Thus, the matrix equivalent of Eq. (1.34) is given by

$$\mathbf{A} = \mathbf{B}^{-1}\mathbf{C} \quad (1.36)$$

Procedures for evaluating the inverse of a square matrix are presented in Examples 1.12 and 1.16.

Matrix factorization refers to the representation of a matrix as the product of two other matrices. For example, a known matrix \mathbf{A} can be represented as the product of two unknown matrices \mathbf{B} and \mathbf{C} . Thus,

$$\mathbf{A} = \mathbf{BC} \quad (1.37)$$

Factorization is not a unique process. There are, in general, an infinite number of matrices \mathbf{B} and \mathbf{C} whose product is \mathbf{A} . A particularly useful factorization for square matrices is

$$\mathbf{A} = \mathbf{LU} \quad (1.38)$$

where \mathbf{L} and \mathbf{U} are lower and upper triangular matrices, respectively. The LU factorization method for solving systems of linear algebraic equations, which is presented in Section 1.4, is based on such a factorization.

A matrix can be *partitioned* by grouping the elements of the matrix into submatrices. These submatrices can then be treated as elements of a smaller matrix. To ensure that the operations of matrix algebra can be applied to the submatrices of two partitioned matrices, the partitioning is generally into square submatrices of equal size. Matrix partitioning is especially convenient when solving systems of algebraic equations that arise in the finite difference solution of systems of differential equations.

1.2.3. Systems of Linear Algebraic Equations

Systems of linear algebraic equations, such as Eq. (1.3), can be expressed very compactly in matrix notation. Thus, Eq. (1.3) can be written as the matrix equation

$$\boxed{\mathbf{Ax} = \mathbf{b}} \quad (1.39)$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix} \quad (1.40)$$

Equation (1.3) can also be written as

$$\sum_{j=1}^n a_{ij}x_j = b_i \quad (i = 1, \dots, n) \quad (1.41)$$

or equivalently as

$$a_{ij}x_j = b_i \quad (i, j = 1, \dots, n) \quad (1.42)$$

where the *summation convention* holds, that is, the repeated index j in Eq. (1.42) is summed over its range, 1 to n . Equation (1.39) will be used throughout this book to represent a system of linear algebraic equations.

There are three so-called *row operations* that are useful when solving systems of linear algebraic equations. They are:

1. Any row (equation) may be multiplied by a constant (a process known as *scaling*).
2. The order of the rows (equations) may be interchanged (a process known as *pivoting*).
3. Any row (equation) can be replaced by a weighted linear combination of that row (equation) with any other row (equation) (a process known as *elimination*).

In the context of the solution of a system of linear algebraic equations, these three row operations clearly do not change the solution. The appearance of the system of equations is obviously changed by any of these row operations, but the solution is unaffected. When solving systems of linear algebraic equations expressed in matrix notation, these row operations apply to the rows of the matrices representing the system of linear algebraic equations.

1.2.4. Determinants

The term *determinant* of a square matrix \mathbf{A} , denoted $\det(\mathbf{A})$ or $|\mathbf{A}|$, refers to both the collection of the elements of the square matrix, enclosed in vertical lines, and the scalar value represented by that array. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \quad (1.43)$$

Only square matrices have determinants.

The scalar value of the determinant of a 2×2 matrix is the product of the elements on the major diagonal minus the product of the elements on the minor diagonal.

Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12} \quad (1.44)$$

The scalar value of the determinant of a 3×3 matrix is composed of the sum of six triple products which can be obtained from the augmented determinant:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{vmatrix} \quad (1.45)$$

The 3×3 determinant is augmented by repeating the first two columns of the determinant on the right-hand side of the determinant. Three triple products are formed, starting with the elements of the first row multiplied by the two remaining elements on the right-

downward-sloping diagonals. Three more triple products are formed, starting with the elements of the third row multiplied by the two remaining elements on the right-upward-sloping diagonals. The value of the determinant is the sum of the first three triple products minus the sum of the last three triple products. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{31}a_{22}a_{13} - a_{32}a_{23}a_{11} - a_{33}a_{21}a_{12} \quad (1.46)$$

Example 1.3. Evaluation of a 3×3 determinant by the diagonal method.

Let's evaluate the determinant of the coefficient matrix of Eq. (1.2) by the diagonal method. Thus,

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \quad (1.47)$$

The augmented determinant is

$$\left| \begin{array}{ccc|cc} 80 & -20 & -20 & 80 & -20 \\ -20 & 40 & -20 & -20 & 40 \\ -20 & -20 & 130 & -20 & -20 \end{array} \right| \quad (1.48)$$

Applying Eq. (1.46) yields

$$\begin{aligned} \det(\mathbf{A}) = |\mathbf{A}| &= (80)(40)(130) + (-20)(-20)(-20) + (-20)(-20)(-20) \\ &\quad - (-20)(40)(-20) - (-20)(-20)(80) \\ &\quad - (130)(-20)(-20) = 416,000 - 8,000 - 8,000 \\ &\quad - 16,000 - 32,000 - 52,000 = 300,000 \end{aligned} \quad (1.49)$$

The diagonal method of evaluating determinants applies only to 2×2 and 3×3 determinants. It is incorrect for 4×4 or larger determinants. In general, the expansion of an $n \times n$ determinant is the sum of all possible products formed by choosing one and only one element from each row and each column of the determinant, with a plus or minus sign determined by the number of permutations of the row and column elements. One formal procedure for evaluating determinants is called *expansion by minors*, or the *method of cofactors*. In this procedure there are $n!$ products to be summed, where each product has n elements. Thus, the expansion of a 10×10 determinant requires the summation of $10!$ products ($10! = 3,628,800$), where each product involves 9 multiplications (the product of 10 elements). This is a total of 32,659,000 multiplications and 3,627,999 additions, not counting the work needed to keep track of the signs. Consequently, the evaluation of determinants by the method of cofactors is impractical, except for very small determinants.

Although the method of cofactors is not recommended for anything larger than a 4×4 determinant, it is useful to understand the concepts involved. The *minor* M_{ij} is the determinant of the $(n - 1) \times (n - 1)$ submatrix of the $n \times n$ matrix \mathbf{A} obtained by deleting the i th row and the j th column. The *cofactor* A_{ij} associated with the minor M_{ij} is defined as

$$A_{ij} = (-1)^{i+j} M_{ij} \quad (1.50)$$

Using cofactors, the determinant of matrix \mathbf{A} is the sum of the products of the elements of any row or column, multiplied by their corresponding cofactors. Thus, expanding across any fixed row i yields

$$\det(\mathbf{A}) = |\mathbf{A}| = \sum_{j=1}^n a_{i,j} A_{i,j} = \sum_{j=1}^n (-1)^{i+j} a_{i,j} M_{i,j} \quad (1.51)$$

Alternatively, expanding down any fixed column j yields

$$\det(\mathbf{A}) = |\mathbf{A}| = \sum_{i=1}^n a_{i,j} A_{i,j} = \sum_{i=1}^n (-1)^{i+j} a_{i,j} M_{i,j} \quad (1.52)$$

Each cofactor expansion reduces the order of the determinant by one, so there are n determinants of order $n - 1$ to evaluate. By repeated application, the cofactors are eventually reduced to 3×3 determinants which can be evaluated by the diagonal method. The amount of work can be reduced by choosing the expansion row or column with as many zeros as possible.

Example 1.4. Evaluation of a 3×3 determinant by the cofactor method.

Let's rework Example 1.3 using the cofactor method. Recall Eq. (1.47):

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \quad (1.53)$$

Evaluate $|\mathbf{A}|$ by expanding across the first row. Thus,

$$|\mathbf{A}| = (80) \begin{vmatrix} 40 & -20 \\ -20 & 130 \end{vmatrix} - (-20) \begin{vmatrix} -20 & -20 \\ -20 & 130 \end{vmatrix} + (-20) \begin{vmatrix} -20 & 40 \\ -20 & -20 \end{vmatrix} \quad (1.54)$$

$$\begin{aligned} |\mathbf{A}| &= 80(5200 + 400) - (-20)(-2600 + 400) + (-20)(400 + 800) \\ &= 384000 - 60000 - 24000 = 300000 \end{aligned} \quad (1.55)$$

If the value of the determinant of a matrix is zero, the matrix is said to be *singular*. A *nonsingular matrix* has a determinant that is nonzero. If any row or column of a matrix has all zero elements, that matrix is singular.

The determinant of a triangular matrix, either upper or lower triangular, is the product of the elements on the major diagonal. It is possible to transform any nonsingular matrix into a triangular matrix in such a way that the value of the determinant is either unchanged or changed in a well-defined way. That procedure is presented in Section 1.3.6. The value of the determinant can then be evaluated quite easily as the product of the elements on the major diagonal.

1.3 DIRECT ELIMINATION METHODS

There are a number of methods for the direct solution of systems of linear algebraic equations. One of the more well-known methods is Cramer's rule, which requires the evaluation of numerous determinants. Cramer's rule is highly inefficient, and thus not recommended. More efficient methods, based on the elimination concept, are recom-

mended. Both Cramer's rule and elimination methods are presented in this section. After presenting *Cramer's rule*, the elimination concept is applied to develop Gauss elimination, Gauss-Jordan elimination, matrix inversion, and determinant evaluation. These concepts are extended to LU factorization and tridiagonal systems of equations in Sections 1.4 and 1.5, respectively.

1.3.1. Cramer's Rule

Although it is not an elimination method, *Cramer's rule* is a direct method for solving systems of linear algebraic equations. Consider the system of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$, which represents n equations. Cramer's rule states that the solution for x_j ($j = 1, \dots, n$) is given by

$$x_j = \frac{\det(\mathbf{A}^j)}{\det(\mathbf{A})} \quad (j = 1, \dots, n) \quad (1.56)$$

where \mathbf{A}^j is the $n \times n$ matrix obtained by replacing column j in matrix \mathbf{A} by the column vector \mathbf{b} . For example, consider the system of two linear algebraic equations:

$$a_{11}x_1 + a_{12}x_2 = b_1 \quad (1.57a)$$

$$a_{21}x_1 + a_{22}x_2 = b_2 \quad (1.57b)$$

Applying Cramer's rule yields

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} \quad \text{and} \quad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} \quad (1.58)$$

The determinants in Eqs. (1.58) can be evaluated by the diagonal method described in Section 1.2.4.

For systems containing more than three equations, the diagonal method presented in Section 1.2.4 does not work. In such cases, the method of cofactors presented in Section 1.2.4 could be used. The number of multiplications and divisions N required by the method of cofactors is $N = (n - 1)(n + 1)!$. For a relatively small system of 10 equations (i.e., $n = 10$), $N = 360,000,000$, which is an enormous number of calculations. For $n = 100$, $N = 10^{157}$, which is obviously ridiculous. The preferred method for evaluating determinants is the elimination method presented in Section 1.3.6. The number of multiplications and divisions required by the elimination method is approximately $N = n^3 + n^2 - n$. Thus, for $n = 10$, $N = 1090$, and for $n = 100$, $N = 1,009,900$. Obviously, the elimination method is preferred.

Example 1.5. Cramer's rule.

Let's illustrate Cramer's rule by solving Eq. (1.2). Thus,

$$80x_1 - 20x_2 - 20x_3 = 20 \quad (1.59a)$$

$$-20x_1 + 40x_2 - 20x_3 = 20 \quad (1.59b)$$

$$-20x_1 - 20x_2 + 130x_3 = 20 \quad (1.59c)$$

First, calculate $\det(\mathbf{A})$. From Example 1.4,

$$\det(\mathbf{A}) = \begin{vmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{vmatrix} = 300,000 \quad (1.60)$$

Next, calculate $\det(\mathbf{A}^1)$, $\det(\mathbf{A}^2)$, and $\det(\mathbf{A}^3)$. For $\det(\mathbf{A}^1)$,

$$\det(\mathbf{A}^1) = \begin{vmatrix} 20 & -20 & -20 \\ 20 & 40 & -20 \\ 20 & -20 & 130 \end{vmatrix} = 180,000 \quad (1.61)$$

In a similar manner, $\det(\mathbf{A}^2) = 300,000$ and $\det(\mathbf{A}^3) = 120,000$. Thus,

$$x_1 = \frac{\det(\mathbf{A}^1)}{\det(\mathbf{A})} = \frac{180,000}{300,000} = 0.60 \quad x_2 = \frac{300,000}{300,000} = 1.00 \quad x_3 = \frac{120,000}{300,000} = 0.40 \quad (1.62)$$

1.3.2. Elimination Methods

Elimination methods solve a system of linear algebraic equations by solving one equation, say the first equation, for one of the unknowns, say x_1 , in terms of the remaining unknowns, x_2 to x_n , then substituting the expression for x_1 into the remaining $n - 1$ equations to determine $n - 1$ equations involving x_2 to x_n . This elimination procedure is performed $n - 1$ times until the last step yields an equation involving only x_n . This process is called *elimination*.

The value of x_n can be calculated from the final equation in the elimination procedure. Then x_{n-1} can be calculated from modified equation $n - 1$, which contains only x_n and x_{n-1} . Then x_{n-2} can be calculated from modified equation $n - 2$, which contains only x_n , x_{n-1} , and x_{n-2} . This procedure is performed $n - 1$ times to calculate x_{n-1} to x_1 . This process is called *back substitution*.

1.3.2.1. Row Operations

The elimination process employs the row operations presented in Section 1.2.3, which are repeated below:

1. Any row (equation) may be multiplied by a constant (scaling).
2. The order of the rows (equations) may be interchanged (pivoting).
3. Any row (equation) can be replaced by a weighted linear combination of that row (equation) with any other row (equation) (elimination).

These row operations, which change the values of the elements of matrix \mathbf{A} and \mathbf{b} , do not change the solution \mathbf{x} to the system of equations.

The first row operation is used to scale the equations, if necessary. The second row operation is used to prevent divisions by zero and to reduce round-off errors. The third row operation is used to implement the systematic elimination process described above.

1.3.2.2. Elimination

Let's illustrate the elimination method by solving Eq. (1.2). Thus,

$$80x_1 - 20x_2 - 20x_3 = 20 \quad (1.63a)$$

$$-20x_1 + 40x_2 - 20x_3 = 20 \quad (1.63b)$$

$$-20x_1 - 20x_2 + 130x_3 = 20 \quad (1.63c)$$

Solve Eq. (1.63a) for x_1 . Thus,

$$x_1 = [20 - (-20)x_2 - (-20)x_3]/80 \quad (1.64)$$

Substituting Eq. (1.64) into Eq. (1.63b) gives

$$-20\{[20 - (-20)x_2 - (-20)x_3]/80\} + 40x_2 - 20x_3 = 20 \quad (1.65)$$

which can be simplified to give

$$35x_2 - 25x_3 = 25 \quad (1.66)$$

Substituting Eq. (1.64) into Eq. (1.63c) gives

$$-20\{[20 - (-20)x_2 - (-20)x_3]/80\} - 20x_2 + 130x_3 = 20 \quad (1.67)$$

which can be simplified to give

$$-25x_2 + 125x_3 = 25 \quad (1.68)$$

Next solve Eq. (1.66) for x_2 . Thus,

$$x_2 = [25 - (-25)x_3]/35 \quad (1.69)$$

Substituting Eq. (1.69) into Eq. (1.68) yields

$$-25\{[25 - (-25)x_3]/35\} + 125x_3 = 25 \quad (1.70)$$

which can be simplified to give

$$\frac{750}{7}x_3 = \frac{300}{7} \quad (1.71)$$

Thus, Eq. (1.63) has been reduced to the upper triangular system

$$80x_1 - 20x_2 - 20x_3 = 20 \quad (1.72a)$$

$$35x_2 - 25x_3 = 25 \quad (1.72b)$$

$$\frac{750}{7}x_3 = \frac{300}{7} \quad (1.72c)$$

which is equivalent to the original equation, Eq. (1.63). This completes the elimination process.

1.3.2.3. Back Substitution

The solution to Eq. (1.72) is accomplished easily by *back substitution*. Starting with Eq. (1.72c) and working backward yields

$$x_3 = 300/750 = 0.40 \quad (1.73a)$$

$$x_2 = [25 - (-25)(0.40)]/35 = 1.00 \quad (1.73b)$$

$$x_1 = [20 - (-20)(1.00) - (-20)(0.40)]/80 = 0.60 \quad (1.73c)$$

Example 1.6. Elimination.

Let's solve Eq. (1.2) by elimination. Recall Eq. (1.2):

$$80x_1 - 20x_2 - 20x_3 = 20 \quad (1.74a)$$

$$-20x_1 + 40x_2 - 20x_3 = 20 \quad (1.74b)$$

$$-20x_1 - 20x_2 + 130x_3 = 20 \quad (1.74c)$$

Elimination involves normalizing the equation above the element to be eliminated by the element immediately above the element to be eliminated, which is called the *pivot element*, multiplying the normalized equation by the element to be eliminated, and subtracting the result from the equation containing the element to be eliminated. This process systematically eliminates terms below the major diagonal, column by column, as illustrated below. The notation $R_i - (\text{em})R_j$ next to the i th equation indicates that the i th equation is to be replaced by the i th equation minus em times the j th equation, where the elimination multiplier, em , is the quotient of the element to be eliminated and the pivot element.

For example, $R_2 - (-20/40)R_1$ beside Eq. (1.75.2) below means replace Eq. (1.75.2) by Eq. (1.75.2) $- (-20/40) \times$ Eq. (1.75.1). The elimination multiplier, $\text{em} = (-20/40)$, is chosen to eliminate the first coefficient in Eq. (1.75.2). All of the coefficients below the major diagonal in the first column are eliminated by linear combinations of each equation with the first equation. Thus,

$$\left[\begin{array}{l} 80x_1 - 20x_2 - 20x_3 = 20 \\ -20x_1 + 40x_2 - 20x_3 = 20 \\ -20x_1 - 20x_2 + 130x_3 = 20 \end{array} \right] \quad (1.75.1)$$

$$R_2 - (-20/40)R_1 \quad (1.75.2)$$

$$\left[\begin{array}{l} 80x_1 - 20x_2 - 20x_3 = 20 \\ 0x_1 + 35x_2 - 25x_3 = 25 \\ -20x_1 - 20x_2 + 135x_3 = 20 \end{array} \right] R_3 - (-20/80)R_1 \quad (1.75.3)$$

The result of this first elimination step is presented in Eq. (1.76), which also shows the elimination operation for the second elimination step. Next the coefficients below the major diagonal in the second column are eliminated by linear combinations with the second equation. Thus,

$$\left[\begin{array}{l} 80x_1 - 20x_2 - 20x_3 = 20 \\ 0x_1 + 35x_2 - 25x_3 = 25 \\ 0x_1 - 25x_2 + 125x_3 = 25 \end{array} \right] R_3 - (-25/35)R_2 \quad (1.76)$$

The result of the second elimination step is presented in Eq. (1.77):

$$\left[\begin{array}{l} 80x_1 - 20x_2 - 20x_3 = 20 \\ 0x_1 + 35x_2 - 25x_3 = 25 \\ 0x_1 + 0x_2 + 750/7x_3 = 300/7 \end{array} \right] \quad (1.77)$$

This process is continued until all the coefficients below the major diagonal are eliminated. In the present example with three equations, this process is now complete, and Eq. (1.77) is the final result. This is the process of elimination.

At this point, the last equation contains only one unknown, x_3 in the present example, which can be solved for. Using that result, the next to last equation can be solved

for x_2 . Using the results for x_3 and x_2 , the first equation can be solved for x_1 . This is the back substitution process. Thus,

$$x_3 = 300/750 = 0.40 \quad (1.78a)$$

$$x_2 = [25 - (-25)(0.40)]/35 = 1.00 \quad (1.78b)$$

$$x_1 = [20 - (-20)(1.00) - (-20)(0.40)]/80 = 0.60 \quad (1.78c)$$

The extension of the elimination procedure to n equations is straightforward.

1.3.2.4. Simple Elimination

The elimination procedure illustrated in Example 1.6 involves manipulation of the coefficient matrix \mathbf{A} and the nonhomogeneous vector \mathbf{b} . Components of the \mathbf{x} vector are fixed in their locations in the set of equations. As long as the columns are not interchanged, column j corresponds to x_j . Consequently, the x_j notation does not need to be carried throughout the operations. Only the numerical elements of \mathbf{A} and \mathbf{b} need to be considered. Thus, the elimination procedure can be simplified by augmenting the \mathbf{A} matrix with the \mathbf{b} vector and performing the row operations on the elements of the augmented \mathbf{A} matrix to accomplish the elimination process, then performing the back substitution process to determine the solution vector. This simplified elimination procedure is illustrated in Example 1.7.

Example 1.7. Simple elimination.

Let's rework Example 1.6 using simple elimination. From Example 1.6, the \mathbf{A} matrix augmented by the \mathbf{b} vector is

$$[\mathbf{A} | \mathbf{b}] = \left[\begin{array}{ccc|c} 80 & -20 & -20 & 20 \\ -20 & 40 & -20 & 20 \\ -20 & -20 & 130 & 20 \end{array} \right] \quad (1.79)$$

Performing the row operations to accomplish the elimination process yields:

$$\left[\begin{array}{ccc|c} 80 & -20 & -20 & 20 \\ -20 & 40 & -20 & 20 \\ -20 & -20 & 130 & 20 \end{array} \right] \begin{matrix} R_2 - (-20/80)R_1 \\ R_3 - (-20/80)R_1 \end{matrix} \quad (1.80)$$

$$\left[\begin{array}{ccc|c} 80 & -20 & -20 & 20 \\ 0 & 35 & -25 & 25 \\ 0 & -25 & 125 & 25 \end{array} \right] \begin{matrix} \\ R_3 - (-25/35)R_2 \end{matrix} \quad (1.81)$$

$$\left[\begin{array}{ccc|c} 80 & -20 & -20 & 20 \\ 0 & 35 & -25 & 25 \\ 0 & 0 & 750/7 & 300/7 \end{array} \right] \rightarrow \begin{matrix} x_1 = [20 - (-20)(1.00) - (-20)(0.40)]/80 \\ = 0.60 \\ x_2 = [25 - (-25)(0.4)]/35 = 1.00 \\ x_3 = 300/750 = 0.40 \end{matrix} \quad (1.82)$$

The back substitution step is presented beside the triangularized augmented \mathbf{A} matrix.

1.3.2.5. Multiple \mathbf{b} Vectors

If more than one \mathbf{b} vector is to be considered, the \mathbf{A} matrix is simply augmented by all of the \mathbf{b} vectors simultaneously. The elimination process is then applied to the multiply augmented \mathbf{A} matrix. Back substitution is then applied one column at a time to the modified \mathbf{b} vectors. A more versatile procedure based on matrix factorization is presented in Section 1.4.

Example 1.8. Simple elimination for multiple \mathbf{b} vectors.

Consider the system of equations presented in Example 1.7 with two \mathbf{b} vectors, $\mathbf{b}_1^T = [20 \ 20 \ 20]$ and $\mathbf{b}_2^T = [20 \ 10 \ 20]$. The doubly augmented \mathbf{A} matrix is

$$[\mathbf{A} \mid \mathbf{b}_1 \ \mathbf{b}_2] = \left[\begin{array}{ccc|cc} 80 & -20 & -20 & 20 & 20 \\ -20 & 40 & -20 & 20 & 10 \\ -20 & -20 & 130 & 20 & 20 \end{array} \right] \quad (1.83)$$

Performing the elimination process yields

$$\left[\begin{array}{ccc|cc} 80 & -20 & -20 & 20 & 20 \\ 0 & 35 & -25 & 25 & 15 \\ 0 & 0 & 750/7 & 300/7 & 250/7 \end{array} \right] \quad (1.84)$$

Performing the back substitution process one column at a time yields

$$\mathbf{x}_1 = \begin{bmatrix} 0.60 \\ 1.00 \\ 0.40 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_2 = \begin{bmatrix} 1/2 \\ 2/3 \\ 1/3 \end{bmatrix} \quad (1.85)$$

1.3.2.6. Pivoting

The element on the major diagonal is called the *pivot* element. The elimination procedure described so far fails immediately if the first pivot element a_{11} is zero. The procedure also fails if any subsequent pivot element $a_{i,i}$ is zero. Even though there may be no zeros on the major diagonal in the original matrix, the elimination process may create zeros on the major diagonal. The simple elimination procedure described so far must be modified to avoid zeros on the major diagonal. This result can be accomplished by rearranging the equations, by interchanging equations (rows) or variables (columns), before each elimination step to put the element of largest magnitude on the diagonal. This process is called *pivoting*. Interchanging both rows and columns is called *full pivoting*. Full pivoting is quite complicated, and thus it is rarely used. Interchanging only rows is called *partial pivoting*. Only partial pivoting is considered in this book.

Pivoting eliminates zeros in the pivot element locations during the elimination process. Pivoting also reduces round-off errors, since the pivot element is a divisor during the elimination process, and division by large numbers introduces smaller round-off errors than division by small numbers. When the procedure is repeated, round-off errors can compound. This problem becomes more severe as the number of equations is increased.

Example 1.9. Elimination with pivoting to avoid zero pivot elements.

Use simple elimination with partial pivoting to solve the following system of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$:

$$\begin{bmatrix} 0 & 2 & 1 \\ 4 & 1 & -1 \\ -2 & 3 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ -3 \\ 5 \end{bmatrix} \quad (1.86)$$

Let's apply the elimination procedure by augmenting \mathbf{A} with \mathbf{b} . The first pivot element is zero, so pivoting is required. The largest number (in magnitude) in the first column under the pivot element occurs in the second row. Thus, interchanging the first and second rows and evaluating the elimination multipliers yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 2 & 1 & 5 \\ -2 & 3 & -3 & 5 \end{array} \right] \begin{array}{l} R_2 - (0/4)R_1 \\ R_3 - (-2/4)R_1 \end{array} \quad (1.87)$$

Performing the elimination operations yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 2 & 1 & 5 \\ 0 & 7/2 & -7/2 & 7/2 \end{array} \right] \quad (1.88)$$

Although the pivot element in the second row is not zero, it is not the largest element in the second column underneath the pivot element. Thus, pivoting is called for again. Note that pivoting is based only on the rows below the pivot element. The rows above the pivot element have already been through the elimination process. Using one of the rows above the pivot element would destroy the elimination already accomplished. Interchanging the second and third rows and evaluating the elimination multiplier yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 7/2 & -7/2 & 7/2 \\ 0 & 2 & 1 & 5 \end{array} \right] \begin{array}{l} \\ R_3 - (4/7)R_2 \end{array} \quad (1.89)$$

Performing the elimination operation yields

$$\left[\begin{array}{ccc|c} 4 & 1 & -1 & -3 \\ 0 & 7/2 & -7/2 & 7/2 \\ 0 & 0 & 3 & 3 \end{array} \right] \rightarrow \begin{array}{l} x_1 = -1 \\ x_2 = 2 \\ x_3 = 1 \end{array} \quad (1.90)$$

The back substitution results are presented beside the triangularized augmented \mathbf{A} matrix.

1.3.2.7. Scaling

The elimination process described so far can incur significant round-off errors when the magnitudes of the pivot elements are smaller than the magnitudes of the other elements in the equations containing the pivot elements. In such cases, scaling is employed to select the pivot elements. After pivoting, elimination is applied to the original equations. Scaling is employed only to select the pivot elements.

Scaled pivoting is implemented as follows. Before elimination is applied to the first column, all of the elements in the first column are scaled (i.e., normalized) by the largest elements in the corresponding rows. Pivoting is implemented based on the scaled elements

in the first column, and elimination is applied to obtain zero elements in the first column below the pivot element. Before elimination is applied to the second column, all of the elements from 2 to n in column 2 are scaled, pivoting is implemented, and elimination is applied to obtain zero elements in column 2 below the pivot element. The procedure is applied to the remaining rows 3 to $n - 1$. Back substitution is then applied to obtain \mathbf{x} .

Example 1.10. Elimination with scaled pivoting to reduce round-off errors.

Let's investigate the advantage of scaling by solving the following linear system:

$$\begin{bmatrix} 3 & 2 & 105 \\ 2 & -3 & 103 \\ 1 & 1 & 3 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 104 \\ 98 \\ 3 \end{bmatrix} \quad (1.91)$$

which has the exact solution $x_1 = -1.0$, $x_2 = 1.0$, and $x_3 = 1.0$. To accentuate the effects of round-off, carry only three significant figures in the calculations. For the first column, pivoting does not appear to be required. Thus, the augmented A matrix and the first set of row operations are given by

$$\left[\begin{array}{ccc|c} 3 & 2 & 105 & 104 \\ 2 & -3 & 103 & 98 \\ 1 & 1 & 3 & 3 \end{array} \right] \begin{array}{l} R_2 - (0.667)R_1 \\ R_3 - (0.333)R_1 \end{array} \quad (1.92)$$

which gives

$$\left[\begin{array}{ccc|c} 3 & 2 & 105 & 104 \\ 0 & -4.33 & 33.0 & 28.6 \\ 0 & 0.334 & -32.0 & -31.6 \end{array} \right] \begin{array}{l} R_3 - (-0.0771)R_2 \end{array} \quad (1.93)$$

Pivoting is not required for the second column. Performing the elimination indicated in Eq. (1.93) yields the triangularized matrix

$$\left[\begin{array}{ccc|c} 3 & 2 & 105 & 104 \\ 0 & -4.33 & 33.0 & 28.9 \\ 0 & 0 & -29.5 & -29.4 \end{array} \right] \quad (1.94)$$

Performing back substitution yields $x_3 = 0.997$, $x_2 = 0.924$, and $x_1 = -0.844$, which does not agree very well with the exact solution $x_3 = 1.0$, $x_2 = 1.0$, and $x_1 = -1.0$. Round-off errors due to the three-digit precision have polluted the solution.

The effects of round-off can be reduced by scaling the equations before pivoting. Since scaling itself introduces round-off, it should be used only to determine if pivoting is required. All calculations should be made with the original unscaled equations.

Let's rework the problem using scaling to determine if pivoting is required. The first step in the elimination procedure eliminates all the elements in the first column under element a_{11} . Before performing that step, let's scale all the elements in column 1 by the largest element in each row. The result is

$$\mathbf{a}_1 = \begin{bmatrix} 3/105 \\ 2/103 \\ 1/3 \end{bmatrix} = \begin{bmatrix} 0.0286 \\ 0.0194 \\ 0.3333 \end{bmatrix} \quad (1.95)$$

where the notation \mathbf{a}_1 denotes the column vector consisting of the scaled elements from the first column of matrix A. The third element of \mathbf{a}_1 is the largest element in \mathbf{a}_1 , which

indicates that rows 1 and 3 of matrix \mathbf{A} should be interchanged. Thus, Eq. (1.91), with the elimination multipliers indicated, becomes

$$\left[\begin{array}{ccc|c} 1 & 1 & 3 & 3 \\ 2 & -3 & 103 & 98 \\ 3 & 2 & 105 & 104 \end{array} \right] R_2 - (2/1)R_1 \quad R_3 - (3/1)R_1 \quad (1.96)$$

Performing the elimination and indicating the next elimination multiplier yields

$$\left[\begin{array}{ccc|c} 1 & 1 & 3 & 3 \\ 0 & -5 & 97 & 92 \\ 0 & -1 & 96 & 95 \end{array} \right] R_3 - (1/5)R_2 \quad (1.97)$$

Scaling the second and third elements of column 2 gives

$$\mathbf{a}_2 = \begin{bmatrix} - \\ -5/97 \\ -1/96 \end{bmatrix} = \begin{bmatrix} - \\ -0.0516 \\ -0.0104 \end{bmatrix} \quad (1.98)$$

Consequently, pivoting is not indicated. Performing the elimination indicated in Eq. (1.97) yields

$$\left[\begin{array}{ccc|c} 1.0 & 1.0 & 3.0 & 3.0 \\ 0.0 & -5.0 & 97.0 & 92.0 \\ 0.0 & 0.0 & 76.6 & 76.6 \end{array} \right] \quad (1.99)$$

Solving Eq. (1.99) by back substitution yields $x_1 = 1.00$, $x_2 = 1.00$, and $x_3 = -1.00$, which is the exact solution. Thus, scaling to determine the pivot element has eliminated the round-off error in this simple example.

1.3.3. Gauss Elimination

The elimination procedure described in the previous section, including scaled pivoting, is commonly called *Gauss elimination*. It is the most important and most useful direct elimination method for solving systems of linear algebraic equations. The Gauss-Jordan method, the matrix inverse method, the LU factorization method, and the Thomas algorithm are all modifications or extensions of the Gauss elimination method. Pivoting is an essential element of Gauss elimination. In cases where all of the elements of the coefficient matrix \mathbf{A} are the same order of magnitude, scaling is not necessary. However, pivoting to avoid zero pivot elements is always required. Scaled pivoting to decrease round-off errors, while very desirable in general, can be omitted at some risk to the accuracy of the solution. When performing Gauss elimination by hand, decisions about pivoting can be made on a case by case basis. When writing a general-purpose computer program to apply Gauss elimination to arbitrary systems of equations, however, scaled pivoting is an absolute necessity. Example 1.10 illustrates the complete Gauss elimination algorithm.

When solving large systems of linear algebraic equations on a computer, the pivoting step is generally implemented by simply keeping track of the order of the rows as they are interchanged without actually interchanging rows, a time-consuming and unnecessary operation. This is accomplished by using an order vector \mathbf{o} whose elements denote the order in which the rows of the coefficient matrix \mathbf{A} and the right-hand-side

vector \mathbf{b} are to be processed. When a row interchange is required, instead of actually interchanging the two rows of elements, the corresponding elements of the order vector are interchanged. The rows of the \mathbf{A} matrix and the \mathbf{b} vector are processed in the order indicated by the order vector \mathbf{o} during both the elimination step and the back substitution step.

As an example, consider the second part of Example 1.10. The order vector has the initial value $\mathbf{o}^T = [1 \ 2 \ 3]$. After scaling, rows 1 and 3 are to be interchanged. Instead of actually interchanging these rows as done in Example 1.10, the corresponding elements of the order vector are changed to yield $\mathbf{o}^T = [3 \ 2 \ 1]$. The first elimination step then uses the third row to eliminate x_1 from the second and first rows. Pivoting is not required for the second elimination step, so the order vector is unchanged, and the second row is used to eliminate x_2 from the first row. Back substitution is then performed in the reverse order of the order vector, \mathbf{o} , that is, in the order 1, 2, 3. This procedure saves computer time for large systems of equations, but at the expense of a slightly more complicated program.

The number of multiplications and divisions required for Gauss elimination is approximately $N = (n^3/3 - n/3)$ for matrix \mathbf{A} and n^2 for each \mathbf{b} . For $n = 10$, $N = 430$, and for $n = 100$, $N = 343,300$. This is a considerable reduction compared to Cramer's rule.

The Gauss elimination procedure, in a format suitable for programming on a computer, is summarized as follows:

1. Define the $n \times n$ coefficient matrix \mathbf{A} , the $n \times 1$ column vector \mathbf{b} , and the $n \times 1$ order vector \mathbf{o} .
2. Starting with column 1, scale column k ($k = 1, 2, \dots, n - 1$) and search for the element of largest magnitude in column k and pivot (interchange rows) to put that coefficient into the $a_{k,k}$ pivot position. This step is actually accomplished by interchanging the corresponding elements of the $n \times 1$ order vector \mathbf{o} .
3. For column k ($k = 1, 2, \dots, n - 1$), apply the elimination procedure to rows i ($i = k + 1, k + 2, \dots, n$) to create zeros in column k below the pivot element, $a_{k,k}$. Do not actually calculate the zeros in column k . In fact, storing the elimination multipliers, $\text{em} = (a_{i,k}/a_{k,k})$, in place of the eliminated elements, $a_{i,k}$, creates the Doolittle LU factorization presented in Section 1.4. Thus,

$$a_{i,j} = a_{i,j} - \left(\frac{a_{i,k}}{a_{k,k}} \right) a_{k,j} \quad (i, j = k + 1, k + 2, \dots, n) \quad (1.100a)$$

$$b_i = b_i - \left(\frac{a_{i,k}}{a_{k,k}} \right) b_k \quad (i = k + 1, k + 2, \dots, n) \quad (1.100b)$$

After step 3 is applied to all k columns, ($k = 1, 2, \dots, n - 1$), the original \mathbf{A} matrix is upper triangular.

4. Solve for \mathbf{x} using back substitution. If more than one \mathbf{b} vector is present, solve for the corresponding \mathbf{x} vectors one at a time. Thus,

$$x_n = \frac{b_n}{a_{n,n}} \quad (1.101a)$$

$$x_i = \frac{b_i - \sum_{j=i+1}^n a_{i,j}x_j}{a_{i,i}} \quad (i = n - 1, n - 2, \dots, 1) \quad (1.101b)$$

1.3.4. Gauss-Jordan Elimination

Gauss-Jordan elimination is a variation of Gauss elimination in which the elements above the major diagonal are eliminated (made zero) as well as the elements below the major diagonal. The \mathbf{A} matrix is transformed to a diagonal matrix. The rows are usually scaled to yield unity diagonal elements, which transforms the \mathbf{A} matrix to the identity matrix, \mathbf{I} . The transformed \mathbf{b} vector is then the solution vector \mathbf{x} . Gauss-Jordan elimination can be used for single or multiple \mathbf{b} vectors.

The number of multiplications and divisions for Gauss-Jordan elimination is approximately $N = (n^3/2 - n/2) + n^2$, which is approximately 50 percent larger than for Gauss elimination. Consequently, Gauss elimination is preferred.

Example 1.11. Gauss-Jordan elimination.

Let's rework Example 1.7 using simple Gauss-Jordan elimination, that is, elimination without pivoting. The augmented \mathbf{A} matrix is [see Eq. (1.79)]

$$\left[\begin{array}{ccc|c} 80 & -20 & -20 & 20 \\ -20 & 40 & -20 & 20 \\ -20 & -20 & 130 & 20 \end{array} \right] R_1/80 \quad (1.102)$$

Scaling row 1 to give $a_{11} = 1$ gives

$$\left[\begin{array}{ccc|c} 1 & -1/4 & -1/4 & 1/4 \\ -20 & 40 & -20 & 20 \\ -20 & -20 & 130 & 20 \end{array} \right] R_2 - (-20)R_1 \quad R_3 - (-20)R_1 \quad (1.103)$$

Applying elimination below row 1 yields

$$\left[\begin{array}{ccc|c} 1 & -1/4 & -1/4 & 1/4 \\ 0 & 35 & -25 & 25 \\ 0 & -25 & 125 & 25 \end{array} \right] R_2/35 \quad (1.104)$$

Scaling row 2 to give $a_{22} = 1$ gives

$$\left[\begin{array}{ccc|c} 1 & -1/4 & -1/4 & 1/4 \\ 0 & 1 & -5/7 & 5/7 \\ 0 & -25 & 125 & 25 \end{array} \right] R_1 - (-1/4)R_2 \quad R_3 - (-25)R_2 \quad (1.105)$$

Applying elimination both above and below row 2 yields

$$\left[\begin{array}{ccc|c} 1 & 0 & -3/7 & 3/7 \\ 0 & 1 & -5/7 & 5/7 \\ 0 & 0 & 750/7 & 300/7 \end{array} \right] R_3/(750/7) \quad (1.106)$$

Scaling row 3 to give $a_{33} = 1$ gives

$$\left[\begin{array}{ccc|c} 1 & 0 & -3/7 & 3/7 \\ 0 & 1 & -5/7 & 5/7 \\ 0 & 0 & 1 & 215 \end{array} \right] R_1 - (-3/7)R_3 \quad R_2 - (-5/7)R_3 \quad (1.107)$$

Applying elimination above row 3 completes the process.

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 0.60 \\ 0 & 1 & 0 & 1.00 \\ 0 & 0 & 1 & 0.40 \end{array} \right] \quad (1.108)$$

The \mathbf{A} matrix has been transformed to the identity matrix \mathbf{I} and the \mathbf{b} vector has been transformed to the solution vector, \mathbf{x} . Thus, $\mathbf{x}^T = [0.60 \quad 1.00 \quad 0.40]$.

The inverse of a square matrix \mathbf{A} is the matrix \mathbf{A}^{-1} such that $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. Gauss-Jordan elimination can be used to evaluate the inverse of matrix \mathbf{A} by augmenting \mathbf{A} with the identity matrix \mathbf{I} and applying the Gauss-Jordan algorithm. The transformed \mathbf{A} matrix is the identity matrix \mathbf{I} , and the transformed identity matrix is the matrix inverse, \mathbf{A}^{-1} . Thus, applying Gauss-Jordan elimination yields

$$\boxed{[\mathbf{A} \mid \mathbf{I}] \rightarrow [\mathbf{I} \mid \mathbf{A}^{-1}]} \quad (1.109)$$

The Gauss-Jordan elimination procedure, in a format suitable for programming on a computer, can be developed to solve Eq. (1.109) by modifying the Gauss elimination procedure presented in Section 1.3.C. Step 1 is changed to augment the $n \times n$ \mathbf{A} matrix with the $n \times n$ identity matrix, \mathbf{I} . Steps 2 and 3 of the procedure are the same. Before performing Step 3, the pivot element is scaled to unity by dividing all elements in the row by the pivot element. Step 3 is expanded to perform elimination above the pivot element as well as below the pivot element. At the conclusion of step 3, the \mathbf{A} matrix has been transformed to the identity matrix, \mathbf{I} , and the original identity matrix, \mathbf{I} , has been transformed to the matrix inverse, \mathbf{A}^{-1} .

Example 1.12. Matrix inverse by Gauss-Jordan elimination.

Let's evaluate the inverse of matrix \mathbf{A} presented in Example 1.7. First, augment matrix \mathbf{A} with the identity matrix, \mathbf{I} . Thus,

$$[\mathbf{A} \mid \mathbf{I}] = \left[\begin{array}{ccc|ccc} 80 & -20 & -20 & 1 & 0 & 0 \\ -20 & 40 & -20 & 0 & 1 & 0 \\ -20 & -20 & 130 & 0 & 0 & 1 \end{array} \right] \quad (1.110)$$

Performing Gauss-Jordan elimination transforms Eq. (1.110) to

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 2/125 & 1/100 & 1/250 \\ 0 & 1 & 0 & 1/100 & 1/30 & 1/150 \\ 0 & 0 & 1 & 1/250 & 1/150 & 7/750 \end{array} \right] \quad (1.111)$$

from which

$$\mathbf{A}^{-1} = \left[\begin{array}{ccc} 2/125 & 1/100 & 1/250 \\ 1/100 & 1/30 & 1/150 \\ 1/250 & 1/150 & 7/750 \end{array} \right] = \left[\begin{array}{ccc} 0.016000 & 0.010000 & 0.004000 \\ 0.010000 & 0.033333 & 0.006667 \\ 0.004000 & 0.006667 & 0.009333 \end{array} \right] \quad (1.112)$$

Multiplying \mathbf{A} times \mathbf{A}^{-1} yields the identity matrix \mathbf{I} , thus verifying the computations.

1.3.5. The Matrix Inverse Method

Systems of linear algebraic equations can be solved using the matrix inverse, \mathbf{A}^{-1} . Consider the general system of linear algebraic equations:

$$\mathbf{Ax} = \mathbf{b} \quad (1.113)$$

Multiplying Eq. (1.113) by \mathbf{A}^{-1} yields

$$\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{Ix} = \mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \quad (1.114)$$

from which

$$\boxed{\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}} \quad (1.115)$$

Thus, when the matrix inverse \mathbf{A}^{-1} of the coefficient matrix \mathbf{A} is known, the solution vector \mathbf{x} is simply the product of the matrix inverse \mathbf{A}^{-1} and the right-hand-side vector \mathbf{b} . Not all matrices have inverses. Singular matrices, that is, matrices whose determinant is zero, do not have inverses. The corresponding system of equations does not have a unique solution.

Example 1.13. The matrix inverse method.

Let's solve the linear system considered in Example 1.7 using the matrix inverse method. The matrix inverse \mathbf{A}^{-1} of the coefficient matrix \mathbf{A} for that linear system is evaluated in Example 1.12. Multiplying \mathbf{A}^{-1} by the vector \mathbf{b} from Example 1.7 gives

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \begin{bmatrix} 2/125 & 1/100 & 1/250 \\ 1/100 & 1/30 & 1/150 \\ 1/250 & 1/150 & 7/750 \end{bmatrix} \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix} \quad (1.116)$$

Performing the matrix multiplication yields

$$x_1 = (2/125)(20) + (1/100)(20) + (1/250)(20) = 0.60 \quad (1.117a)$$

$$x_2 = (1/100)(20) + (1/30)(20) + (1/150)(20) = 1.00 \quad (1.117b)$$

$$x_3 = (1/250)(20) + (1/150)(20) + (7/750)(20) = 0.04 \quad (1.117c)$$

Thus, $x^T = [0.60 \quad 1.00 \quad 0.04]$.

1.3.6. Determinants

The evaluation of determinants by the cofactor method is discussed in Section 1.2.4 and illustrated in Example 1.4. Approximately $N = (n - 1)n!$ multiplications are required to evaluate the determinant of an $n \times n$ matrix by the cofactor method. For $n = 10$, $N = 32,659,000$. Evaluation of the determinants of large matrices by the cofactor method is prohibitively expensive, if not impossible. Fortunately, determinants can be evaluated much more efficiently by a variation of the elimination method.

First, consider the matrix \mathbf{A} expressed in upper triangular form:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ \dots & \dots & \dots & \cdots & \dots \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix} \quad (1.118)$$

Expanding the determinant of \mathbf{A} by cofactors down the first column gives a_{11} times the $(n - 1) \times (n - 1)$ determinant having a_{22} as its first element in its first column, with the remaining elements in its first column all zero. Expanding that determinant by cofactors down its first column yields a_{22} times the $(n - 2) \times (n - 2)$ determinant having a_{33} as its first element in its first column with the remaining elements in its first column all zero. Continuing in this manner yields the result that the determinant of an upper triangular matrix (or a lower triangular matrix) is simply the product of the elements on the major diagonal. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = \prod_{i=1}^n a_{i,i} \quad (1.119)$$

where the \prod notation denotes the product of the $a_{i,i}$. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = a_{11}a_{22} \cdots a_{nn} \quad (1.120)$$

This result suggests the use of elimination to triangularize a general square matrix, then to evaluate its determinant using Eq. (1.119). This procedure works exactly as stated if no pivoting is used. When pivoting is used, the value of the determinant is changed, but in a predictable manner, so elimination can also be used with pivoting to evaluate determinants. The row operations must be modified as follows to use elimination for the evaluation of determinants.

1. Multiplying a row by a constant multiplies the determinant by that constant.
2. Interchanging any two rows changes the sign of the determinant. Thus, an even number of row interchanges does not change the sign of the determinant, whereas an odd number of row interchanges does change the sign of the determinant.
3. Any row may be added to the multiple of any other row without changing the value of the determinant.

The modified elimination method based on the above row operations is an efficient way to evaluate the determinant of a matrix. The number of multiplications required is approximately $N = n^3 + n^2 - n$, which is orders and orders of magnitude less effort than the $N = (n - 1)n!$ multiplications required by the cofactor method.

Example 1.14. Evaluation of a 3×3 determinant by the elimination method.

Let's rework Example 1.4 using the elimination method. Recall Eq. (1.53):

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \quad (1.121)$$

From Example 1.7, after Gauss elimination, matrix \mathbf{A} becomes

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \quad (1.122)$$

There are no row interchanges or multiplications of the matrix by scalars in this example. Thus,

$$\det(\mathbf{A}) = |\mathbf{A}| = (80)(35)(750/7) = 300,000 \quad (1.123)$$

1.4 LU FACTORIZATION

Matrices (like scalars) can be *faktored* into the product of two other matrices in an infinite number of ways. Thus,

$$\mathbf{A} = \mathbf{BC} \quad (1.124)$$

When \mathbf{B} and \mathbf{C} are lower triangular and upper triangular matrices, respectively, Eq. (1.124) becomes

$$\mathbf{A} = \mathbf{LU} \quad (1.125)$$

Specifying the diagonal elements of either \mathbf{L} or \mathbf{U} makes the factoring unique. The procedure based on unity elements on the major diagonal of \mathbf{L} is called the Doolittle method. The procedure based on unity elements on the major diagonal of \mathbf{U} is called the Crout method.

Matrix factoring can be used to reduce the work involved in Gauss elimination when multiple unknown \mathbf{b} vectors are to be considered. In the Doolittle LU method, this is accomplished by defining the elimination multipliers, e_{mj} , determined in the elimination step of Gauss elimination as the elements of the \mathbf{L} matrix. The \mathbf{U} matrix is defined as the upper triangular matrix determined by the elimination step of Gauss elimination. In this manner, multiple \mathbf{b} vectors can be processed through the elimination step using the \mathbf{L} matrix and through the back substitution step using the elements of the \mathbf{U} matrix.

Consider the linear system, $\mathbf{Ax} = \mathbf{b}$. Let \mathbf{A} be factored into the product \mathbf{LU} , as illustrated in Eq. (1.125). The linear system becomes

$$\mathbf{LUx} = \mathbf{b} \quad (1.126)$$

Multiplying Eq. (1.126) by \mathbf{L}^{-1} gives

$$\mathbf{L}^{-1}\mathbf{LUx} = \mathbf{L}^{-1}\mathbf{b} \quad (1.127)$$

The last two terms in Eq. (1.127) give

$$\mathbf{Ux} = \mathbf{L}^{-1}\mathbf{b} \quad (1.128)$$

Define the vector \mathbf{b}' as follows:

$$\mathbf{b}' = \mathbf{L}^{-1}\mathbf{b} \quad (1.129)$$

Multiplying Eq. (1.129) by \mathbf{L} gives

$$\mathbf{Lb}' = \mathbf{LL}^{-1}\mathbf{b} = \mathbf{Ib} = \mathbf{b} \quad (1.130)$$

Equating the first and last terms in Eq. (1.130) yields

$$\boxed{\mathbf{L}\mathbf{b}' = \mathbf{b}} \quad (1.131)$$

Substituting Eq. (1.129) into Eq. (1.128) yields

$$\boxed{\mathbf{Ux} = \mathbf{b}'} \quad (1.132)$$

Equation (1.131) is used to transform the \mathbf{b} vector into the \mathbf{b}' vector, and Eq. (1.132) is used to determine the solution vector \mathbf{x} . Since Eq. (1.131) is lower triangular, forward substitution (analogous to back substitution presented earlier) is used to solve for \mathbf{b}' . Since Eq. (1.132) is upper triangular, back substitution is used to solve for \mathbf{x} .

In the *Doolittle LU method*, the \mathbf{U} matrix is the upper triangular matrix obtained by Gauss elimination. The \mathbf{L} matrix is the lower triangular matrix containing the elimination multipliers, em , obtained in the Gauss elimination process as the elements below the diagonal, with unity elements on the major diagonal. Equation (1.131) applies the steps performed in the triangularization of \mathbf{A} to \mathbf{U} to the \mathbf{b} vector to transform \mathbf{b} to \mathbf{b}' . Equation (1.132) is simply the back substitution step of the Gauss elimination method. Consequently, once \mathbf{L} and \mathbf{U} have been determined, any \mathbf{b} vector can be considered at any later time, and the corresponding solution vector \mathbf{x} can be obtained simply by solving Eqs. (1.131) and (1.132), in that order. The number of multiplicative operations required for each \mathbf{b} vector is n^2 .

Example 1.15. The Doolittle LU method.

Let's solve Example 1.7 using the Doolittle LU method. The first step is to determine the \mathbf{L} and \mathbf{U} matrices. The \mathbf{U} matrix is simply the upper triangular matrix determined by the Gauss elimination procedure in Example 1.7. The \mathbf{L} matrix is simply the record of the elimination multipliers, em , used to transform \mathbf{A} to \mathbf{U} . These multipliers are the numbers in parentheses in the row operations indicated in Eqs. (1.80) and (1.81) in Example 1.7. Thus, \mathbf{L} and \mathbf{U} are given by

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \quad (1.133)$$

Consider the first \mathbf{b} vector from Example 1.8: $\mathbf{b}_1^T = [20 \ 20 \ 20]$. Equation (1.131) gives

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix} \quad (1.134)$$

Performing forward substitution yields

$$b'_1 = 20 \quad (1.135a)$$

$$b'_2 = 20 - (-1/4)(20) = 25 \quad (1.135b)$$

$$b'_3 = 20 - (-1/4)(20) - (-5/7)(25) = 300/7 \quad (1.135c)$$

The \mathbf{b}' vector is simply the transformed \mathbf{b} vector determined in Eq. (1.82). Equation (1.132) gives

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 25 \\ 300/7 \end{bmatrix} \quad (1.136)$$

Performing back substitution yields $x_1^T = [0.60 \quad 1.00 \quad 0.40]$. Repeating the process for $\mathbf{b}_2^T = [20 \quad 10 \quad 20]$ yields

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 20 \end{bmatrix} \begin{array}{l} b'_1 = 20 \\ b'_2 = 15 \\ b'_3 = 250/7 \end{array} \quad (1.137)$$

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 15 \\ 250/7 \end{bmatrix} \begin{array}{l} x_1 = 1/2 \\ x_2 = 2/3 \\ x_3 = 1/3 \end{array} \quad (1.138)$$

When pivoting is used with LU factorization, it is necessary to keep track of the row order, for example, by an order vector \mathbf{o} . When the rows of \mathbf{A} are interchanged during the elimination process, the corresponding elements of the order vector \mathbf{o} are interchanged. When a new \mathbf{b} vector is considered, it is processed in the order corresponding to the elements of the order vector \mathbf{o} .

The major advantage of LU factorization methods is their efficiency when multiple unknown \mathbf{b} vectors must be considered. The number of multiplications and divisions required by the complete Gauss elimination method is $N = (n^3/3 - n/3) + n^2$. The forward substitution step required to solve $\mathbf{L}\mathbf{b}' = \mathbf{b}$ requires $N = n^2/2 - n/2$ multiplicative operations, and the back substitution step required to solve $\mathbf{U}\mathbf{x} = \mathbf{b}'$ requires $N = n^2/2 + n/2$ multiplicative operations. Thus, the total number of multiplicative operations required by LU factorization, after \mathbf{L} and \mathbf{U} have been determined, is n^2 , which is much less work than required by Gauss elimination, especially for large systems.

The Doolittle LU method, in a format suitable for programming for a computer, is summarized as follows:

1. Perform steps 1, 2, and 3 of the Gauss elimination procedure presented in Section 1.3.3. Store the pivoting information in the order vector \mathbf{o} . Store the row elimination multipliers, \mathbf{em} , in the locations of the eliminated elements. The results of this step are the \mathbf{L} and \mathbf{U} matrices.
2. Compute the \mathbf{b}' vector in the order of the elements of the order vector \mathbf{o} using forward substitution:

$$b'_i = b_i - \sum_{k=1}^{i-1} l_{i,k} b'_k \quad (i = 2, 3, \dots, n) \quad (1.139)$$

where $l_{i,k}$ are the elements of the \mathbf{L} matrix.

3. Compute the \mathbf{x} vector using back substitution:

$$x_i = b'_i - \sum_{k=i+1}^n u_{i,k} x_k / u_{i,i} \quad (i = n-1, n-2, \dots, 1) \quad (1.140)$$

where $u_{i,k}$ and $u_{i,i}$ are elements of the \mathbf{U} matrix.

As a final application of LU factorization, it can be used to evaluate the inverse of matrix \mathbf{A} , that is, \mathbf{A}^{-1} . The matrix inverse is calculated in a column by column manner using unit vectors for the right-hand-side vector \mathbf{b} . Thus, if $\mathbf{b}_1^T = [1 \ 0 \ \dots \ 0]$, \mathbf{x}_1 will be the first column of \mathbf{A}^{-1} . The succeeding columns of \mathbf{A}^{-1} are calculated by letting $\mathbf{b}_2^T = [0 \ 1 \ \dots \ 0]$, $\mathbf{b}_3^T = [0 \ 0 \ 1 \ \dots \ 0]$, etc., and $\mathbf{b}_n^T = [0 \ 0 \ \dots \ 1]$. The number of multiplicative operations for each column is n^2 . There are n columns, so the total number of multiplicative operations is n^3 . The number of multiplicative operations required to determine \mathbf{L} and \mathbf{U} are $(n^3/3 - n/3)$. Thus, the total number of multiplicative operations required is $4n^3/3 - n/3$, which is smaller than the $3n^3/2 - n/2$ operations required by the Gauss-Jordan method.

Example 1.16. Matrix inverse by the Doolittle LU method.

Let's evaluate the inverse of matrix \mathbf{A} presented in Example 1.7 by the Doolittle LU method:

$$\mathbf{A} = \begin{bmatrix} 80 & -20 & -20 \\ -20 & 40 & -20 \\ -20 & -20 & 130 \end{bmatrix} \quad (1.141)$$

Evaluate the \mathbf{L} and \mathbf{U} matrices by Doolittle LU factorization. Thus,

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \quad (1.142)$$

Let $\mathbf{b}_1^T = [1 \ 0 \ 0]$. Then, $\mathbf{L}\mathbf{b}_1' = \mathbf{b}_1$ gives

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \mathbf{b}'_1 = \begin{bmatrix} 1 \\ 1/4 \\ 3/7 \end{bmatrix} \quad (1.143a)$$

Solve $\mathbf{Ux} = \mathbf{b}'_1$ to determine \mathbf{x}_1 . Thus,

$$\begin{bmatrix} 80 & -20 & -20 \\ 0 & 35 & -25 \\ 0 & 0 & 750/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1/4 \\ 3/7 \end{bmatrix} \rightarrow \mathbf{x}_1 = \begin{bmatrix} 2/125 \\ 1/100 \\ 1/250 \end{bmatrix} \quad (1.143b)$$

where \mathbf{x}_1 is the first column of \mathbf{A}^{-1} . Letting $\mathbf{b}_2^T = [0 \ 1 \ 0]$ gives $\mathbf{x}_2^T = [1/100 \ 1/30 \ 1/150]$, and letting $\mathbf{b}_3^T = [0 \ 0 \ 1]$ gives $\mathbf{x}_3^T = [1/250 \ 1/150 \ 7/750]$. Thus, \mathbf{A}^{-1} is given by

$$\begin{aligned} \mathbf{A}^{-1} &= [\mathbf{x}_1 \ \mathbf{x}_2 \ \mathbf{x}_3] = \begin{bmatrix} 2/125 & 1/100 & 1/250 \\ 1/100 & 1/30 & 1/150 \\ 1/250 & 1/150 & 7/750 \end{bmatrix} \\ &= \begin{bmatrix} 0.016000 & 0.010000 & 0.004000 \\ 0.01000 & 0.033333 & 0.006667 \\ 0.004000 & 0.006667 & 0.009333 \end{bmatrix} \end{aligned} \quad (1.143c)$$

which is the same result obtained by Gauss-Jordan elimination in Example 1.12.

1.5 TRIDIAGONAL SYSTEMS OF EQUATIONS

When a large system of linear algebraic equations has a special pattern, such as a tridiagonal pattern, it is usually worthwhile to develop special methods for that unique pattern. There are a number of direct elimination methods for solving systems of linear algebraic equations which have special patterns in the coefficient matrix. These methods are generally very efficient in computer time and storage. Such methods should be considered when the coefficient matrix fits the required pattern, and when computer storage and/or execution time are important. One algorithm that deserves special attention is the algorithm for tridiagonal matrices, often referred to as the Thomas (1949) algorithm. Large tridiagonal systems arise naturally in a number of problems, especially in the numerical solution of differential equations by implicit methods. Consequently, the Thomas algorithm has found a large number of applications.

To derive the Thomas algorithm, let's apply the Gauss elimination procedure to a tridiagonal matrix \mathbf{T} , modifying the procedure to eliminate all unnecessary computations involving zeros. Consider the matrix equation:

$$\boxed{\mathbf{T}\mathbf{x} = \mathbf{b}} \quad (1.144)$$

where \mathbf{T} is a tridiagonal matrix. Thus,

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & \cdots & 0 & 0 & 0 \\ \dots & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & \cdots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \quad (1.145)$$

Since all the elements of column 1 below row 2 are already zero, the only element to be eliminated in row 2 is a_{21} . Thus, replace row 2 by $R_2 - (a_{21}/a_{11})R_1$. Row 2 becomes

$$[0 \quad a_{22} - (a_{21}/a_{11})a_{12} \quad a_{23} \quad 0 \quad 0 \quad \cdots \quad 0 \quad 0 \quad 0] \quad (1.146)$$

Similarly, only a_{32} in column 2 must be eliminated from row 3, only a_{43} in column 3 must be eliminated from row 4, etc. The eliminated element itself does not need to be calculated. In fact, storing the elimination multipliers, $em = (a_{21}/a_{11})$, etc., in place of the eliminated elements allows this procedure to be used as an LU factorization method. Only the diagonal element in each row is affected by the elimination. Elimination in rows 2 to n is accomplished as follows:

$$a_{i,i} = a_{i,i} - (a_{i,i-1}/a_{i-1,i-1})a_{i-1,i} \quad (i = 2, \dots, n) \quad (1.147)$$

Thus, the elimination step involves only $2n$ multiplicative operations to place \mathbf{T} in upper triangular form.

The elements of the \mathbf{b} vector are also affected by the elimination process. The first element b_1 is unchanged, The second element b_2 becomes

$$b_2 = b_2 - (a_{21}/a_{11})b_1 \quad (1.148)$$

Subsequent elements of the \mathbf{b} vector are changed in a similar manner. Processing the \mathbf{b} vector requires only one multiplicative operation, since the elimination multiplier,

$\mathbf{em} = (a_{21}/a_{11})$, is already calculated. Thus, the total process of elimination, including the operation on the \mathbf{b} vector, requires only $3n$ multiplicative operations.

The $n \times n$ tridiagonal matrix \mathbf{T} can be stored as an $n \times 3$ matrix \mathbf{A}' since there is no need to store the zeros. The first column of matrix \mathbf{A}' , elements $a'_{i,1}$, corresponds to the subdiagonal of matrix \mathbf{T} , elements $a_{i,i-1}$. The second column of matrix \mathbf{A}' , elements $a'_{i,2}$, corresponds to the diagonal elements of matrix \mathbf{T} , elements $a_{i,i}$. The third column of matrix \mathbf{A}' , elements $a'_{i,3}$, corresponds to the superdiagonal of matrix \mathbf{T} , elements $a_{i,i+1}$. The elements $a'_{1,1}$ and $a'_{n,3}$ do not exist. Thus,

$$\mathbf{A}' = \begin{bmatrix} & a'_{1,2} & a'_{1,3} \\ a'_{2,1} & a'_{2,2} & a'_{2,3} \\ a'_{3,1} & a'_{3,2} & a'_{3,3} \\ \dots & \dots & \dots \\ a'_{n-1,1} & a'_{n-1,2} & a'_{n-1,3} \\ a'_{n,1} & a'_{n,2} & — \end{bmatrix} \quad (1.149)$$

When the elements of column 1 of matrix A' are eliminated, that is, the elements $a'_{i,1}$, the elements of column 2 of matrix A' become

$$a'_{1,2} = a'_{1,2} \quad (1.150a)$$

$$d'_{i,2} = d'_{i,2} - (d'_{i,1}/d'_{i-1,2})d'_{i-1,3} \quad (i = 2, 3, \dots, n) \quad (1.150b)$$

The **b** vector is modified as follows:

$$b_1 = b_1 \quad (1.151a)$$

$$b_i = b_i - (a'_{i,1}/a'_{i-1,2})b_{i-1} \quad (i = 2, 3, \dots, n) \quad (1.151b)$$

After $a'_{i,2}$ ($i = 2, 3, \dots, n$) and \mathbf{b} are evaluated, the back substitution step is as follows:

$$x_n = b_n/a'_{n,2} \quad (1.152a)$$

$$x_i = (b_i - a'_{i,3}x_{i+1})/a'_{i,2} \quad (i = n-1, n-2, \dots, 1) \quad (1.152b)$$

Example 1.17. The Thomas algorithm.

Let's solve the tridiagonal system of equations obtained in Example 8.4, Eq. (8.54). In that example, the finite difference equation

$$T_{i-1} - (2 + \alpha^2 \Delta x^2) T_i + T_{i+1} = 0 \quad (1.153)$$

is solved for $\alpha = 4.0$ and $\Delta x = 0.125$, for which $(2 + \alpha^2 \Delta x^2) = 2.25$, for $i = 2, \dots, 8$, with $T_1 = 0.0$ and $T_9 = 100.0$. Writing Eq. (1.153) in the form of the $n \times 3$ matrix \mathbf{A}' (where the temperatures T_i of Example 8.4 correspond to the elements of the \mathbf{x} vector) yields

The major diagonal terms (the center column of the \mathbf{A}' matrix) are transformed according to Eq. (1.150). Thus, $a'_{1,2} = -2.25$ and $a'_{2,2}$ is given by

$$a'_{2,2} = a'_{2,2} - (a'_{2,1}/a'_{1,2})a'_{1,3} = -2.25 - [1.0/(-2.25)](1.0) = -1.805556 \quad (1.155)$$

The remaining elements of column 2 are processed in the same manner. The \mathbf{A}' matrix after elimination is presented in Eq. (1.157), where the elimination multipliers are presented in parentheses in column 1. The \mathbf{b} vector is transformed according to Eq. (1.151). Thus, $b_1 = 0.0$, and

$$b_2 = b_2 - (a'_{2,1}/a'_{1,2})b_1 = 0.0 - [1.0/(-2.25)](0.0) = 0.0 \quad (1.156)$$

The remaining elements of \mathbf{b} are processed in the same manner. The results are presented in Eq. (1.157). For this particular \mathbf{b} vector, where elements b_1 to b_{n-1} are all zero, the \mathbf{b} vector does not change. This is certainly not the case in general. The final result is:

$$\mathbf{A}' = \begin{bmatrix} - & -2.250000 & 1.0 \\ (-0.444444) & -1.805556 & 1.0 \\ (-0.553846) & -1.696154 & 1.0 \\ (-0.589569) & -1.660431 & 1.0 \\ (-0.602253) & -1.647747 & 1.0 \\ (-0.606889) & -1.643111 & 1.0 \\ (-0.608602) & -1.641398 & - \end{bmatrix} \quad \text{and} \quad \mathbf{b}' = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ -100.0 \end{bmatrix} \quad (1.157)$$

The solution vector is computed using Eq. (1.152). Thus,

$$x_7 = b_7/a'_{7,2} = (-100)/(-1.641398) = 60.923667 \quad (1.158a)$$

$$\begin{aligned} x_6 &= (b_6 - a'_{6,3}x_7)/a'_{6,2} = [0 - (1.0)(60.923667)]/(-1.643111) \\ &= 37.078251 \end{aligned} \quad (1.158b)$$

Processing the remaining rows yields the solution vector:

$$\mathbf{x} = \begin{bmatrix} 1.966751 \\ 4.425190 \\ 7.989926 \\ 13.552144 \\ 22.502398 \\ 37.078251 \\ 60.923667 \end{bmatrix} \quad (1.158c)$$

Equation (1.158c) is the solution presented in Table 8.9.

Pivoting destroys the tridiagonality of the system of linear algebraic equations, and thus cannot be used with the Thomas algorithm. Most large tridiagonal systems which represent real physical problems are diagonally dominant, so pivoting is not necessary.

The number of multiplicative operations required by the elimination step is $N = 2n - 3$ and the number of multiplicative operations required by the back substitution step is $N = 3n - 2$. Thus, the total number of multiplicative operations is $N = 5n - 4$ for the complete Thomas algorithm. If the \mathbf{T} matrix is constant and multiple \mathbf{b} vectors are to be considered, only the back substitution step is required once the \mathbf{T} matrix has been factored into \mathbf{L} and \mathbf{U} matrices. In that case, $N = 3n - 2$ for subsequent \mathbf{b} vectors. The advantages of the Thomas algorithm are quite apparent when compared with either the Gauss elimination method, for which $N = (n^3/3 - n/3) + n^2$, or the Doolittle LU method, for

which $N = n^2 - n/2$, for each \mathbf{b} vector after the first one. The Thomas algorithm, in a format suitable for programming for a computer, is summarized as follows:

1. Store the $n \times n$ tridiagonal matrix \mathbf{T} in the $n \times 3$ matrix \mathbf{A}' . The right-hand-side vector \mathbf{b} is an $n \times 1$ column vector.
2. Compute the $a'_{i,2}$ terms from Eq. (1.150). Store the elimination multipliers, $\text{em} = a'_{i,1}/a'_{i-1,2}$, in place of $a'_{i,1}$.
3. Compute the b_i terms from Eq. (1.151).
4. Solve for x_i by back substitution using Eq. (1.152).

An extended form of the Thomas algorithm can be applied to *block tridiagonal matrices*, in which the elements of \mathbf{T} are partitioned into submatrices having similar patterns. The solution procedure is analogous to that just presented for scalar elements, except that matrix operations are employed on the submatrix elements.

An algorithm similar to the Thomas algorithm can be developed for other special types of systems of linear algebraic equations. For example, a pentadiagonal system of linear algebraic equations is illustrated in Example 8.6.

1.6. PITFALLS OF ELIMINATION METHODS

All nonsingular systems of linear algebraic equations have a solution. In theory, the solution can always be obtained by Gauss elimination. However, there are two major pitfalls in the application of Gauss elimination (or its variations): (a) the presence of round-off errors, and (b) ill-conditioned systems. Those pitfalls are discussed in this section. The effects of round-off can be reduced by a procedure known as iterative improvement, which is presented at the end of this section.

1.6.1. Round-Off Errors

Round-off errors occur when exact infinite precision numbers are approximated by finite precision numbers. In most computers, single precision representation of numbers typically contains about 7 significant digits, double precision representation typically contains about 14 significant digits, and quad precision representation typically contains about 28 significant digits. The effects of round-off errors are illustrated in the following example.

Example 1.18. Effects of round-off errors.

Consider the following system of linear algebraic equations:

$$0.0003x_1 + 3x_2 = 1.0002 \quad (1.159a)$$

$$x_1 + x_2 = 1 \quad (1.159b)$$

Solve Eq. (1.159) by Gauss elimination. Thus,

$$\left[\begin{array}{cc|c} 0.0003 & 3 & 1.0002 \\ 1 & 1 & 1 \end{array} \right] R_2 - R_1/0.0003 \quad (1.160a)$$

$$\left[\begin{array}{cc|c} 0.0003 & 3 & 1.0002 \\ 0 & -9999 & 1 - \frac{1.0002}{0.0003} \end{array} \right] \quad (1.160b)$$

Table 1.1. Solution of Eq. (1.162)

Precision	x_2	x_1
3	0.333	3.33
4	0.3332	1.333
5	0.33333	0.70000
6	0.333333	0.670000
7	0.3333333	0.6670000
8	0.33333333	0.66670000

The exact solution of Eq. (1.160) is

$$x_2 = \frac{1 - \frac{1.0002}{0.0003}}{-9999} = \frac{\frac{0.0003 - 1.0002}{0.0003}}{-9999} = \frac{-0.9999}{-9999} = \frac{1}{3} \quad (1.161a)$$

$$x_1 = \frac{1.0002 - 3x_2}{0.0003} = \frac{1.0002 - 3(1/3)}{0.0003} = \frac{0.0002}{0.0003} = \frac{2}{3} \quad (1.161b)$$

Let's solve Eq. (1.161) using finite precision arithmetic with two to eight significant figures. Thus,

$$x_2 = \frac{1 - \frac{1.0002}{0.0003}}{-9999} \quad \text{and} \quad x_1 = \frac{1.0002 - 3x_2}{0.0003} \quad (1.162)$$

The results are presented in Table 1.1. The algorithm is clearly performing very poorly.

Let's rework the problem by interchanging rows 1 and 2 in Eq. (1.159). Thus,

$$x_1 + x_2 = 1 \quad (1.163a)$$

$$0.0003x_1 + 3x_2 = 1.0002 \quad (1.163b)$$

Solve Eq. (1.163) by Gauss elimination. Thus,

$$\left[\begin{array}{cc|c} 1 & 1 & 1 \\ 0.0003 & 3 & 1.0002 \end{array} \right] R_2 - 0.0003R_1 \quad (1.164a)$$

$$\left[\begin{array}{cc|c} 1 & 1 & 1 \\ 0 & 2.9997 & 0.9999 \end{array} \right] \quad (1.164b)$$

$$x_2 = \frac{0.9999}{2.9997} \quad \text{and} \quad x_1 = 1 - x_2 \quad (1.164c)$$

Let's solve Eq. (1.164c) using finite precision arithmetic. The results are presented in Table 1.2. These results clearly demonstrate the benefits of pivoting.

Table 1.2. Solution of Eq. (1.164c)

Precision	x_2	x_1
3	0.333	0.667
4	0.3333	0.6667
5	0.33333	0.66667

Round-off errors can never be completely eliminated. However, they can be minimized by using high precision arithmetic and pivoting.

1.6.2. System Condition

All well-posed nonsingular numerical problems have an exact solution. In theory, the exact solution can always be obtained using fractions or infinite precision numbers (i.e., an infinite number of significant digits). However, all practical calculations are done with finite precision numbers which necessarily contain round-off errors. The presence of round-off errors alters the solution of the problem.

A well-conditioned problem is one in which a small change in any of the elements of the problem causes only a small change in the solution of the problem.

An ill-conditioned problem is one in which a small change in any of the elements of the problem causes a large change in the solution of the problem. Since ill-conditioned systems are extremely sensitive to small changes in the elements of the problem, they are also extremely sensitive to round-off errors.

Example 1.19. System condition.

Let's illustrate the behavior of an ill-conditioned system by the following problem:

$$x_1 + x_2 = 2 \quad (1.165a)$$

$$x_1 + 1.0001x_2 = 2.0001 \quad (1.165b)$$

Solve Eq. (1.165) by Gauss elimination. Thus,

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 1 & 1.0001 & 2.0001 \end{array} \right] R_2 - R_1 \quad (1.166a)$$

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 0.0001 & 0.0001 \end{array} \right] \quad (1.166b)$$

Solving Eq. (1.166b) yields $x_2 = 1$ and $x_1 = 1$.

Consider the following slightly modified form of Eq. (1.165) in which a_{22} is changed slightly from 1.0001 to 0.9999:

$$x_1 + x_2 = 2 \quad (1.167a)$$

$$x_1 + 0.9999x_2 = 2.0001 \quad (1.167b)$$

Solving Eq. (1.167) by Gauss elimination gives

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 1 & 0.9999 & 2.0001 \end{array} \right] R_2 - R_1 \quad (1.168a)$$

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & -0.0001 & 0.0001 \end{array} \right] \quad (1.168b)$$

Solving Eq. (1.168b) yields $x_2 = -1$ and $x_1 = 3$, which is greatly different from the solution of Eq. (1.165).

Consider another slightly modified form of Eq. (1.165) in which b_2 is changed slightly from 2.0001 to 2:

$$x_1 + x_2 = 2 \quad (1.169a)$$

$$x_1 + 1.0001x_2 = 2 \quad (1.169b)$$

Solving Eq. (1.169) by Gauss elimination gives

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 1 & 1.0001 & 2 \end{array} \right] R_2 - R_1 \quad (1.170a)$$

$$\left[\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 0.0001 & 0 \end{array} \right] \quad (1.170b)$$

Solving Eq. (1.170) yields $x_2 = 0$ and $x_1 = 2$, which is greatly different from the solution of Eq. (1.165).

This problem illustrates that very small changes in any of the elements of \mathbf{A} or \mathbf{b} can cause extremely large changes in the solution, \mathbf{x} . Such a system is ill-conditioned.

With infinite precision arithmetic, ill-conditioning is not a problem. However, with finite precision arithmetic, round-off errors effectively change the elements of \mathbf{A} and \mathbf{b} slightly, and if the system is ill-conditioned, large changes (i.e., errors) can occur in the solution. Assuming that scaled pivoting has been performed, the only possible remedy to ill-conditioning is to use higher precision arithmetic.

There are several ways to check a matrix \mathbf{A} for ill-conditioning. If the magnitude of the determinant of the matrix is small, the matrix may be ill-conditioned. However, this is not a foolproof test. The inverse matrix \mathbf{A}^{-1} can be calculated, and \mathbf{AA}^{-1} can be computed and compared to \mathbf{I} . Similarly, $(\mathbf{A}^{-1})^{-1}$ can be computed and compared to \mathbf{A} . A close comparison in either case suggests that matrix \mathbf{A} is well-conditioned. A poor comparison suggests that the matrix is ill-conditioned. Some of the elements of \mathbf{A} and/or \mathbf{b} can be changed slightly, and the solution repeated. If a drastically different solution is obtained, the matrix is probably ill-conditioned. None of these approaches is foolproof, and none give a quantitative measure of ill-conditioning. The surest way to detect ill-conditioning is to evaluate the condition number of the matrix, as discussed in the next subsection.

1.6.3. Norms and the Condition Number

The problems associated with an ill-conditioned system of linear algebraic equations are illustrated in the previous discussion. In the following discussion, ill-conditioning is quantified by the condition number of a matrix, which is defined in terms of the norms of the matrix and its inverse. Norms and the condition number are discussed in this section.

1.6.3.1. Norms

The measure of the magnitude of \mathbf{A} , \mathbf{x} , or \mathbf{b} is called its *norm* and denoted by $\|\mathbf{A}\|$, $\|\mathbf{x}\|$, and $\|\mathbf{b}\|$, respectively. Norms have the following properties:

$$\|\mathbf{A}\| > 0 \quad (1.171a)$$

$$\|\mathbf{A}\| = 0 \quad \text{only if } \mathbf{A} = \mathbf{0} \quad (1.171b)$$

$$\|k\mathbf{A}\| = |k|\|\mathbf{A}\| \quad (1.171c)$$

$$\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\| \quad (1.171d)$$

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\| \quad (1.171e)$$

The norm of a scalar is its absolute value. Thus, $\|k\| = |k|$. There are several definitions of the norm of a vector. Thus,

$$\|\mathbf{x}\|_1 = \sum |x_i| \quad \text{Sum of magnitudes} \quad (1.172a)$$

$$\|\mathbf{x}\|_2 = \|\mathbf{x}\|_e = \left(\sum x_i^2 \right)^{1/2} \quad \text{Euclidean norm} \quad (1.172b)$$

$$\|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i| \quad \text{Maximum magnitude norm} \quad (1.172c)$$

The *Euclidean norm* is the length of the vector in n -space.

In a similar manner, there are several definitions of the norm of a matrix. Thus,

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{i,j}| \quad \text{Maximum column sum} \quad (1.173a)$$

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{i,j}| \quad \text{Maximum row sum} \quad (1.173b)$$

$$\|\mathbf{A}\|_2 = \min \lambda_i \quad (\text{eigenvalue}) \quad \text{Spectral norm} \quad (1.173c)$$

$$\|\mathbf{A}\|_e = \left(\sum_{i=1}^n \sum_{j=1}^n a_{i,j}^2 \right)^{1/2} \quad \text{Euclidean norm} \quad (1.173d)$$

1.6.3.2. Condition Number

The *condition number* of a system is a measure of the sensitivity of the system to small changes in any of its elements. Consider a system of linear algebraic equations:

$$\mathbf{Ax} = \mathbf{b} \quad (1.174)$$

For Eq. (1.174),

$$\|\mathbf{b}\| \leq \|\mathbf{A}\|\|\mathbf{x}\| \quad (1.175)$$

Consider a slightly modified form of Eq. (1.174) in which \mathbf{b} is altered by $\delta\mathbf{b}$, which causes a change in the solution $\delta\mathbf{x}$. Thus,

$$\mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b} \quad (1.176)$$

Subtracting Eq. (1.174) from Eq. (1.176) gives

$$\mathbf{A} \delta\mathbf{x} = \delta\mathbf{b} \quad (1.177)$$

Solving Eq. (1.177) for $\delta\mathbf{x}$ gives

$$\delta\mathbf{x} = \mathbf{A}^{-1} \delta\mathbf{b} \quad (1.178)$$

For Eq. (1.178),

$$\|\delta\mathbf{x}\| \leq \|\mathbf{A}^{-1}\| \|\delta\mathbf{b}\| \quad (1.179)$$

Multiplying the left-hand and right-hand sides of Eqs. (1.175) and (1.179) gives

$$\|\mathbf{b}\| \|\delta\mathbf{x}\| \leq \|\mathbf{A}\| \|\mathbf{x}\| \|\mathbf{A}^{-1}\| \|\delta\mathbf{b}\| \quad (1.180)$$

Dividing Eqs. (1.180) by $\|\mathbf{b}\| \|\mathbf{x}\|$ yields

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} = C(\mathbf{A}) \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \quad (1.181)$$

where $C(\mathbf{A})$ is the *condition number* of matrix \mathbf{A} :

$$C(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\| \quad (1.182)$$

Equation (1.182) determines the sensitivity of the solution, $\|\delta\mathbf{x}\|/\|\mathbf{x}\|$, to changes in the vector \mathbf{b} , $\|\delta\mathbf{b}\|/\|\mathbf{b}\|$. The sensitivity is determined directly by the value of the condition number $C(\mathbf{A})$. Small values of $C(\mathbf{A})$, of the order of unity, show a small sensitivity of the solution to changes in \mathbf{b} . Such a problem is well-conditioned. Large values of $C(\mathbf{A})$ show a large sensitivity of the solution to changes in \mathbf{b} . Such a problem is ill-conditioned.

It can be shown by a similar analysis that perturbing the matrix \mathbf{A} instead of the vector \mathbf{b} gives

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x} + \delta\mathbf{x}\|} \leq C(\mathbf{A}) \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} \quad (1.183)$$

The use of the condition number is illustrated in Example 1.20.

Example 1.20. Norms and condition numbers.

Consider the coefficient matrix of Eq. (1.159):

$$\mathbf{A} = \begin{bmatrix} 0.0003 & 3 \\ 1 & 1 \end{bmatrix} \quad (1.184)$$

The Euclidian norm of matrix \mathbf{A} is

$$\|\mathbf{A}\|_e = [(0.0003)^2 + 3^2 + 1^2 + 1^2]^{1/2} = 3.3166 \quad (1.185)$$

The inverse of matrix \mathbf{A} is

$$\mathbf{A}^{-1} = \begin{bmatrix} \frac{1}{(0.0003)9,999} & \frac{10,000}{9,999} \\ \frac{1}{(0.0003)9,999} & -\frac{1}{9,999} \end{bmatrix} \quad (1.186)$$

The Euclidian norm of \mathbf{A}^{-1} is $\|\mathbf{A}^{-1}\|_e = 1.1057$. Thus, the condition number of matrix \mathbf{A} is

$$C(\mathbf{A}) = \|\mathbf{A}\|_e \|\mathbf{A}^{-1}\|_e = 3.3166(1.1057) = 3.6672 \quad (1.187)$$

This relatively small condition number shows that matrix \mathbf{A} is well-conditioned. As shown in Section 1.6.1, Eq. (1.159) is sensitive to the precision of the arithmetic (i.e., round-off effects), even though it is well-conditioned. This is a precision problem, not a condition problem.

Consider the coefficient matrix of Eq. (1.165):

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix} \quad (1.188)$$

The Euclidean norm of matrix \mathbf{A} is $\|\mathbf{A}\|_e = 2.00005$. The inverse of matrix \mathbf{A} is

$$\mathbf{A}^{-1} = \begin{bmatrix} 10,001 & -10,000 \\ -10,000 & 10,000 \end{bmatrix} \quad (1.189)$$

The Euclidean norm of \mathbf{A}^{-1} is $\|\mathbf{A}^{-1}\|_e = 20,000.5$. Thus, the condition number of matrix \mathbf{A} is

$$C(\mathbf{A}) = \|\mathbf{A}\|_e \|\mathbf{A}^{-1}\|_e = (2.00005)20,000.5 = 40,002.0 \quad (1.190)$$

This large condition number shows that matrix \mathbf{A} is ill-conditioned.

1.6.4. Iterative Improvement

In all direct elimination methods, the effects of round-off propagate as the solution progresses through the system of equations. The accumulated effect of round-off is *round-off error* in the computed values. Round-off errors in any calculation can be decreased by using higher precision (i.e., more significant digits) arithmetic. Round-off errors in direct elimination methods of solving systems of linear algebraic equations are minimized by using scaled pivoting. Further reduction in round-off errors can be achieved by a procedure known as *iterative improvement*.

Consider a system of linear algebraic equations:

$$\mathbf{Ax} = \mathbf{b} \quad (1.191)$$

Solving Eq. (1.191) by a direct elimination method yields $\tilde{\mathbf{x}}$, where $\tilde{\mathbf{x}}$ differs from the exact solution \mathbf{x} by the error $\delta\mathbf{x}$, where $\tilde{\mathbf{x}} = \mathbf{x} + \delta\mathbf{x}$. Substituting $\tilde{\mathbf{x}}$ into Eq. (1.191) gives

$$\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{Ax} + \mathbf{A}\delta\mathbf{x} = \mathbf{b} + \delta\mathbf{b} \quad (1.192)$$

From the first and last terms in Eq. (1.192), $\delta\mathbf{b}$, is given by

$$\delta\mathbf{b} = \mathbf{A}\tilde{\mathbf{x}} - \mathbf{b} \quad (1.193)$$

Subtracting $\mathbf{A}\tilde{\mathbf{x}} = \mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \mathbf{A}\delta\mathbf{x}$ into Eq. (1.193) gives a system of linear algebraic equations for $\delta\mathbf{x}$. Thus,

$$\boxed{\mathbf{A}\delta\mathbf{x} = \delta\mathbf{b}} \quad (1.194)$$

Equation (1.194) can be solved for $\delta\mathbf{x}$, which can be added to $\tilde{\mathbf{x}}$ to give an improved approximation to \mathbf{x} . The procedure can be repeated (i.e., iterated) if necessary. A convergence check on the value of $\delta\mathbf{x}$ can be used to determine if the procedure should be repeated. If the procedure is iterated, LU factorization should be used to reduce the

computational effort since matrix \mathbf{A} is constant. Equation (1.194) should be solved with higher precision than the precision used in the solution of Eq. (1.191).

1.7 ITERATIVE METHODS

For many large systems of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$, the coefficient matrix \mathbf{A} is extremely sparse. That is, most of the elements of \mathbf{A} are zero. If the matrix is diagonally dominant [see Eq. (1.15)], it is generally more efficient to solve such systems of linear algebraic equations by iterative methods than by direct elimination methods. Three iterative methods are presented in this section: *Jacobi iteration*, *Gauss-Seidel iteration*, and *successive-over-relaxation (SOR)*.

Iterative methods begin by assuming an initial solution vector $\mathbf{x}^{(0)}$. The initial solution vector is used to generate an improved solution vector $\mathbf{x}^{(1)}$ based on some strategy for reducing the difference between $\mathbf{x}^{(0)}$ and the actual solution vector \mathbf{x} . This procedure is repeated (i.e., iterated) to convergence. The procedure is convergent if each iteration produces approximations to the solution vector that approach the exact solution vector as the number of iterations increases.

Iterative methods do not converge for all sets of equations, nor for all possible arrangements of a particular set of equations. *Diagonal dominance* is a sufficient condition for convergence of Jacobi iteration, Gauss-Seidel iteration, and SOR, for any initial solution vector. Diagonal dominance is defined by Eq. (1.15). Some systems that are not diagonally dominant can be rearranged (i.e., by row interchanges) to make them diagonally dominant. Some systems that are not diagonally dominant may converge for certain initial solution vectors, but convergence is not assured. Iterative methods should not be used for systems of linear algebraic equations that cannot be made diagonally dominant.

When repeated application of an iterative method produces insignificant changes in the solution vector, the procedure should be terminated. In other words, the algorithm is repeated (iterated) until some specified convergence criterion is achieved. Convergence is achieved when some measure of the relative or absolute change in the solution vector is less than a specified convergence criterion. The number of iterations required to achieve convergence depends on:

1. The dominance of the diagonal coefficients. As the diagonal dominance increases, the number of iterations required to satisfy the convergence criterion decreases.
2. The method of iteration used.
3. The initial solution vector.
4. The convergence criterion specified.

1.7.1. The Jacobi Iteration Method

Consider the general system of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$, written in index notation:

$$\sum_{j=1}^n a_{i,j}x_j = b_i \quad (i = 1, 2, \dots, n) \tag{1.195}$$

In Jacobi iteration, each equation of the system is solved for the component of the solution vector associated with the diagonal element, that is, x_i . Thus,

$$x_i = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j - \sum_{j=i+1}^n a_{i,j}x_j \right) \quad (i = 1, 2, \dots, n) \quad (1.196)$$

An initial solution vector $\mathbf{x}^{(0)}$ is chosen. The superscript in parentheses denotes the iteration number, with zero denoting the initial solution vector. The initial solution vector $\mathbf{x}^{(0)}$ is substituted into Eq. (1.196) to yield the first improved solution vector $\mathbf{x}^{(1)}$. Thus,

$$x_i^{(1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{(0)} - \sum_{j=i+1}^n a_{i,j}x_j^{(0)} \right) \quad (i = 1, 2, \dots, n) \quad (1.197)$$

This procedure is repeated (i.e., iterated) until some convergence criterion is satisfied. The Jacobi algorithm for the general iteration step (k) is:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{(k)} - \sum_{j=i+1}^n a_{i,j}x_j^{(k)} \right) \quad (i = 1, 2, \dots, n) \quad (1.198)$$

An equivalent, but more convenient, form of Eq. (1.198) can be obtained by adding and subtracting $x_i^{(k)}$ from the right-hand side of Eq. (1.198) to yield

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^n a_{i,j}x_j^{(k)} \right) \quad (i = 1, 2, \dots, n) \quad (1.199)$$

Equation (1.199) is generally written in the form

$$x_i^{(k+1)} = x_i^{(k)} + \frac{R_i^{(k)}}{a_{i,i}} \quad (i = 1, 2, \dots, n) \quad (1.200a)$$

$$R_i^{(k)} = b_i - \sum_{j=1}^n a_{i,j}x_j^{(k)} \quad (i = 1, 2, \dots, n) \quad (1.200b)$$

where the term $R_i^{(k)}$ is called the *residual* of equation i . The residuals $R_i^{(k)}$ are simply the net values of the equations evaluated for the approximate solution vector $\mathbf{x}^{(k)}$.

The Jacobi method is sometimes called the method of simultaneous iteration because all values of x_i are iterated simultaneously. That is, all values of $x_i^{(k+1)}$ depend only on the values of $x_i^{(k)}$. The order of processing the equations is immaterial.

Example 1.21. The Jacobi iteration method.

To illustrate the Jacobi iteration method, let's solve the following system of linear algebraic equations:

$$\begin{bmatrix} 4 & -1 & 0 & 1 & 0 \\ -1 & 4 & -1 & 0 & 1 \\ 0 & -1 & 4 & -1 & 0 \\ 1 & 0 & -1 & 4 & -1 \\ 0 & 1 & 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \quad (1.201)$$

Table 1.3. Solution by the Jacobi Iteration Method

k	x_1	x_2	x_3	x_4	x_5
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	25.000000	25.000000	25.000000	25.000000	25.000000
2	25.000000	31.250000	37.500000	31.250000	25.000000
3	25.000000	34.375000	40.625000	34.375000	25.000000
4	25.000000	35.156250	42.187500	35.156250	25.000000
5	25.000000	35.546875	42.578125	35.546875	25.000000
...
16	25.000000	35.714284	42.857140	35.714284	25.000000
17	25.000000	35.714285	42.857142	35.714285	25.000000
18	25.000000	35.714285	42.857143	35.714285	25.000000

Equation (1.201), when expanded, becomes

$$4x_1 - x_2 + x_4 = 100 \quad (1.202.1)$$

$$-x_1 + 4x_2 - x_3 + x_5 = 100 \quad (1.202.2)$$

$$-x_2 + 4x_3 - x_4 = 100 \quad (1.202.3)$$

$$x_1 - x_3 + 4x_4 - x_5 = 100 \quad (1.202.4)$$

$$x_2 - x_4 + 4x_5 = 100 \quad (1.202.5)$$

Equation (1.202) can be rearranged to yield expressions for the residuals, R_i . Thus,

$$R_1 = 100 - 4x_1 + x_2 - x_4 \quad (1.203.1)$$

$$R_2 = 100 + x_1 - 4x_2 + x_3 - x_5 \quad (1.203.2)$$

$$R_3 = 100 + x_2 - 4x_3 + x_4 \quad (1.203.3)$$

$$R_4 = 100 - x_1 + x_3 - 4x_4 + x_5 \quad (1.203.4)$$

$$R_5 = 100 - x_2 + x_4 - 4x_5 \quad (1.203.5)$$

To initiate the solution, let $\mathbf{x}^{(0)T} = [0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.0]$. Substituting these values into Eq. (1.203) gives $R_i^{(0)} = 100.0$ ($i = 1, \dots, 5$). Substituting these values into Eq. (1.202a) gives $x_1^{(1)} = x_2^{(1)} = x_3^{(1)} = x_4^{(1)} = x_5^{(1)} = 25.0$. The procedure is then repeated with these values to obtain $\mathbf{x}^{(2)}$, etc.

The first and subsequent iterations are summarized in Table 1.3. Due to the symmetry of the coefficient matrix \mathbf{A} and the symmetry of the \mathbf{b} vector, $x_1 = x_5$ and $x_2 = x_4$. The calculations were carried out on a 13-digit precision computer and iterated until all $|\Delta x_i|$ changed by less than 0.000001 between iterations, which required 18 iterations.

1.7.2. Accuracy and Convergence of Iterative Methods

All nonsingular systems of linear algebraic equations have an exact solution. In principle, when solved by direct methods, the exact solution can be obtained. However, all real calculations are performed with finite precision numbers, so round-off errors pollute the

solution. Round-off errors can be minimized by pivoting, but even the most careful calculations are subject to the round-off characteristics of the computing device (i.e., hand computation, hand calculator, personal computer, work station, or mainframe computer).

Iterative methods are less susceptible to round-off errors than direct elimination methods for three reasons: (a) The system of equations is diagonally dominant, (b) the system of equations is typically sparse, and (c) each iteration through the system of equations is independent of the round-off errors of the previous iteration.

When solved by iterative methods, the exact solution of a system of linear algebraic equations is approached asymptotically as the number of iterations increases. When the number of iterations increases without bound, the numerical solution yields the exact solution within the round-off limit of the computing device. Such solutions are said to be correct to machine accuracy. In most practical solutions, machine accuracy is not required. Thus, the iterative process should be terminated when some type of accuracy criterion (or criteria) has been satisfied. In iterative methods, the term *accuracy* refers to the number of significant figures obtained in the calculations, and the term *convergence* refers to the point in the iterative process when the desired accuracy is obtained.

1.7.2.1. Accuracy

The *accuracy* of any approximate method is measured in terms of the *error* of the method. There are two ways to specify error: *absolute error* and *relative error*. Absolute error is defined as

$$\text{Absolute error} = \text{approximate value} - \text{exact value} \quad (1.204)$$

and relative error is defined as

$$\text{Relative error} = \frac{\text{absolute error}}{\text{exact value}} \quad (1.205)$$

Relative error can be stated directly or as a percentage.

Consider an iterative calculation for which the desired absolute error is ± 0.001 . If the exact solution is 100.000, then the approximate value is 100.000 ± 0.001 , which has five significant digits. However, if the exact solution is 0.001000, then the approximate value is 0.001000 ± 0.001 , which has no significant digits. This example illustrates the danger of using absolute error as an accuracy criterion. When the magnitude of the exact solution is known, an absolute accuracy criterion can be specified to yield a specified number of significant digits in the approximate solution. Otherwise, a relative accuracy criterion is preferable.

Consider an iterative calculation for which the desired relative error is ± 0.00001 . If the exact solution is 100.000, then the absolute error must be $100.000 \times (\pm 0.00001) = \pm 0.001$ to satisfy the relative error criterion. This yields five significant digits in the approximate value. If the exact solution is 0.001000, then the absolute error must be $0.001000 \times (\pm 0.00001) = \pm 0.00000001$ to satisfy the relative error criterion. This yields five significant digits in the approximate solution. A relative error criterion yields the same number of significant figures in the approximate value, regardless of the magnitude of the exact solution.

1.7.2.2. Convergence

Convergence of an iterative procedure is achieved when the desired accuracy criterion (or criteria) is satisfied. Convergence criteria can be specified in terms of absolute error or relative error. Since the exact solution is unknown, the error at any step in the iterative

process is based on the change in the quantity being calculated from one step to the next. Thus, for the iterative solution of a system of linear algebraic equations, the error, $\Delta x_i = x_i^{(k+1)} - x_i^{\text{exact}}$, is approximated by $x_i^{(k+1)} - x_i^{(k)}$. The error can also be specified by the magnitudes of the residuals R_i . When the exact answer (or the exact answer to machine accuracy) is obtained, the residuals are all zero. At each step in the iterative procedure, some of the residuals may be near zero while others are still quite large. Therefore, care is needed to ensure that the desired accuracy of the complete system of equations is achieved.

Let ε be the magnitude of the convergence tolerance. Several convergence criteria are possible. For an absolute error criterion, the following choices are possible:

$$|(\Delta x_i)_{\max}| \leq \varepsilon \quad \sum_{i=1}^n |\Delta x_i| \leq \varepsilon \quad \text{or} \quad \left[\sum_{i=1}^n (\Delta x_i)^2 \right]^{1/2} \leq \varepsilon \quad (1.206)$$

For a relative error criterion, the following choices are possible:

$$\left| \frac{(\Delta x_i)_{\max}}{x_i} \right| \leq \varepsilon \quad \sum_{i=1}^n \left| \frac{\Delta x_i}{x_i} \right| \leq \varepsilon \quad \text{or} \quad \left[\sum_{i=1}^n \left(\frac{\Delta x_i}{x_i} \right)^2 \right]^{1/2} \leq \varepsilon \quad (1.207)$$

The concepts of accuracy and convergence discussed in this section apply to all iterative procedures, not just the iterative solution of a system of linear algebraic equations. They are relevant to the solution of eigenvalue problems (Chapter 2), to the solution of nonlinear equations (Chapter 3), etc.

1.7.3. The Gauss-Seidel Iteration Method

In the Jacobi method, all values of $\mathbf{x}^{(k+1)}$ are based on $\mathbf{x}^{(k)}$. The Gauss-Seidel method is similar to the Jacobi method, except that the most recently computed values of all x_i are used in all computations. In brief, as better values of x_i are obtained, use them immediately. Like the Jacobi method, the Gauss-Seidel method requires diagonal dominance to ensure convergence. The Gauss-Seidel algorithm is obtained from the Jacobi algorithm, Eq. (1.198), by using $x_j^{(k+1)}$ values in the summation from $j = 1$ to $i - 1$ (assuming the sweeps through the equations proceed from $i = 1$ to n). Thus,

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^n a_{i,j} x_j^{(k)} \right) \quad (i = 1, 2, \dots, n) \quad (1.208)$$

Equation (1.208) can be written in terms of the residuals R_i by adding and subtracting $x_i^{(k)}$ from the right-hand side of the equation and rearranging to yield

$$x_i^{(k+1)} = x_i^{(k)} + \frac{R_i^{(k)}}{a_{i,i}} \quad (i = 1, 2, \dots, n) \quad (1.209)$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^n a_{i,j} x_j^{(k)} \quad (i = 1, 2, \dots, n) \quad (1.210)$$

The Gauss-Seidel method is sometimes called the method of successive iteration because the most recent values of all x_i are used in all the calculations. Gauss-Seidel iteration generally converges faster than Jacobi iteration.

Table 1.4. Solution by the Gauss-Seidel Iteration Method

k	x_1	x_2	x_3	x_4	x_5
0	0.000000	0.000000	0.000000	0.000000	
1	25.000000	31.250000	32.812500	26.953125	23.925781
2	26.074219	33.740234	40.173340	34.506226	25.191498
3	24.808502	34.947586	42.363453	35.686612	25.184757
4	24.815243	35.498485	42.796274	35.791447	25.073240
5	24.926760	35.662448	42.863474	35.752489	25.022510
...
13	25.000002	35.714287	42.857142	35.714285	25.999999
14	25.000001	35.714286	42.857143	35.714285	25.000000
15	25.000000	35.714286	42.857143	35.714286	25.000000

Example 1.22. The Gauss-Seidel iteration method.

Let's rework the problem presented in Example 1.21 using Gauss-Seidel iteration. The residuals are given by Eq. (1.210). Substituting the initial solution vector, $\mathbf{x}^{(0)T} = [0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.0]$, into Eq. (1.210.1) gives $R_1^{(0)} = 100.0$. Substituting that result into Eq. (1.209.1) gives $x_1^{(1)} = 25.0$. Substituting $\mathbf{x}^T = [25.0 \ 0.0 \ 0.0 \ 0.0 \ 0.0]$ into Eq. (1.210.2) gives

$$R_2^{(1)} = (100.0 + 25.0) = 125.0 \quad (1.211a)$$

Substituting this result into Eq. (1.209.2) yields

$$x_2^{(1)} = 0.0 + \frac{125.0}{4} = 31.25 \quad (1.211b)$$

Continuing in this manner yields $R_3^{(1)} = 131.250$, $x_3^{(1)} = 32.81250$, $R_4^{(1)} = 107.81250$, $x_4^{(1)} = 26.953125$, $R_5^{(1)} = 95.703125$, and $x_5^{(1)} = 23.925781$.

The first and subsequent iterations are summarized in Table 1.4. The intermediate iterates are no longer symmetrical as they were in Example 1.21. The calculations were carried out on a 13-digit precision computer and iterated until all $|\Delta x_i|$ changed by less than 0.000001 between iterations, which required 15 iterations, which is three less than required by the Jacobi method in Example 1.21.

1.7.4. The Successive-Over-Relaxation (SOR) Method

Iterative methods are frequently referred to as relaxation methods, since the iterative procedure can be viewed as relaxing $\mathbf{x}^{(0)}$ to the exact value \mathbf{x} . Historically, the method of relaxation, or just the term relaxation, refers to a specific procedure attributed to Southwell (1940). *Southwell's relaxation method* embodies two procedures for accelerating the convergence of the basic iteration scheme. First, the relaxation order is determined by visually searching for the residual of greatest magnitude, $|R_i|_{\max}$, and then relaxing the corresponding equation by calculating a new value of x_i so that $(R_i)_{\max} = 0.0$. This changes the other residuals that depend on x_i . As the other residuals are relaxed, the value of R_i moves away from zero. The procedure is applied repetitively until all the residuals satisfy the convergence criterion (or criteria).

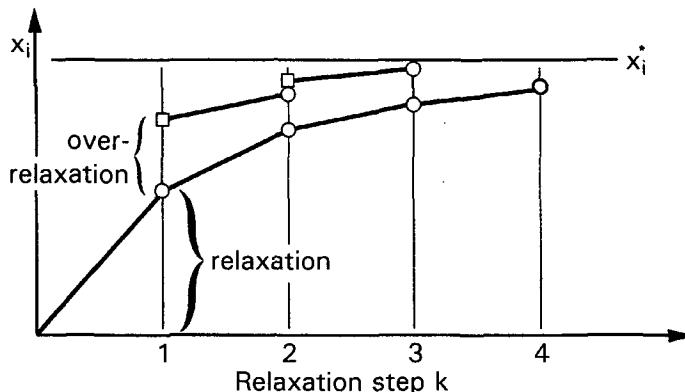


Figure 1.4 Over-relaxation.

Southwell observed that in many cases the changes in x_i from iteration to iteration were always in the same directions. Consequently, over-correcting (i.e., over-relaxing) the values of x_i by the right amount accelerates convergence. This procedure is illustrated in Figure 1.4.

Southwell's method is quite efficient for hand calculation. However, the search for the largest residual is inefficient for computer application, since it can take almost as much computer time to search for the largest residual as it does to make a complete pass through the iteration procedure. On the other hand, the over-relaxation concept is easy to implement on the computer and is very effective in accelerating the convergence rate of the Gauss-Seidel method.

The Gauss-Seidel method can be modified to include over-relaxation simply by multiplying the residual $R_i^{(k)}$ in Eq. (1.209), by the over-relaxation factor, ω . Thus, the *successive-over-relaxation method* is given by

$$x_i^{(k+1)} = x_i^{(k)} + \omega \frac{R_i^{(k)}}{a_{i,i}} \quad (i = 1, 2, \dots, n) \quad (1.212)$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i}^n a_{i,j} x_j^{(k)} \quad (i = 1, 2, \dots, n) \quad (1.213)$$

When $\omega = 1.0$, Eq. (1.212) yields the Gauss-Seidel method. When $1.0 < \omega < 2.0$, the system of equations is over-relaxed. Over-relaxation is appropriate for systems of linear algebraic equations. When $\omega < 1.0$, the system of equations is under-relaxed. Under-relaxation is appropriate when the Gauss-Seidel algorithm causes the solution vector to overshoot and move farther away from the exact solution. This behavior is generally associated with the iterative solution of systems of nonlinear algebraic equations. The iterative method diverges if $\omega \geq 2.0$. The relaxation factor does not change the final solution since it multiplies the residual R_i , which is zero when the final solution is reached. The major difficulty with the over-relaxation method is the determination of the best value for the over-relaxation factor, ω . Unfortunately, there is not a good general method for determining the optimum over-relaxation factor, ω_{opt} .

The optimum value of the over-relaxation factor ω_{opt} depends on the size of the system of equations (i.e., the number of equations) and the nature of the equations (i.e., the

strength of the diagonal dominance, the structure of the coefficient matrix, etc.). As a general rule, larger values of ω_{opt} are associated with larger systems of equations. In Section 9.6, Eqs. (9.51) and (9.52), a procedure is described for estimating ω_{opt} for the system of equations obtained when solving the Laplace equation in a rectangular domain with Dirichlet boundary conditions. In general, one must resort to numerical experimentation to determine ω_{opt} . In spite of this inconvenience, it is almost always worthwhile to search for a near optimum value of ω if a system of equations is to be solved many times. In some problems, the computation time can be reduced by factors as large as 10 to 50. For serious calculations with a large number of equations, the potential is too great to ignore.

Example 1.23. The SOR method.

To illustrate the SOR method, let's rework the problem presented in Example 1.22 using $\omega = 1.10$. The residuals are given by Eq. (1.213). Substituting the initial solution vector, $\mathbf{x}^{(0)T} = [0.0 \ 0.0 \ 0.0 \ 0.0 \ 0.0]$, into Eq. (1.213.1) gives $R_1^{(0)} = 100.0$. Substituting that value into Eq. (1.212.1) with $\omega = 1.10$ gives

$$x_1^{(1)} = 0.0 + 1.10 \frac{100.0}{4} = 27.500000 \quad (1.214a)$$

Substituting $\mathbf{x}^T = [27.50 \ 0.0 \ 0.0 \ 0.0 \ 0.0]$ into Eq. (1.213.2) gives

$$R_2^{(0)} = (100.0 + 27.50) = 127.50 \quad (1.214b)$$

Substituting this result into Eq. (1.212.2) gives

$$x_2^{(1)} = 0.0 + 1.10 \frac{127.50}{4} = 35.062500 \quad (1.214c)$$

Continuing in this manner yields the results presented in Table 1.5.

The first and subsequent iterations are summarized in Table 1.5. The calculations were carried out on a 13-digit precision computer and iterated until all $|\Delta x_i|$ changed by less than 0.000001 between iterations, which required 13 iterations, which is 5 less than required by the Jacobi method and 2 less than required by the Gauss-Seidel method. The value of over-relaxation is modest in this example. Its value becomes more significant as the number of equations increases.

Table 1.5. Solution by the SOR Method

k	x_1	x_2	x_3	x_4	x_5
0	0.000000	0.000000	0.000000	0.000000	0.000000
1	27.500000	35.062500	37.142188	30.151602	26.149503
2	26.100497	34.194375	41.480925	35.905571	25.355629
3	24.419371	35.230346	42.914285	35.968342	25.167386
4	24.855114	35.692519	42.915308	35.790750	25.010375
5	24.987475	35.726188	42.875627	35.717992	24.996719
...
11	24.999996	35.714285	42.857145	35.714287	25.000000
12	25.000000	35.714286	42.857143	35.714286	25.000000
13	25.000000	35.714286	42.857143	35.714286	25.000000

Table 1.6. Number of Iterations k as a Function of ω

ω	k	ω	k	ω	k
1.00	15	1.06	13	1.12	13
1.01	14	1.07	13	1.13	13
1.02	14	1.08	13	1.14	13
1.03	14	1.09	13	1.15	14
1.04	14	1.10	13		
1.05	13	1.11	13		

The optimum value of ω can be determined by experimentation. If a problem is to be worked only once, that procedure is not worthwhile. However, if a problem is to be worked many times with the same \mathbf{A} matrix for many different \mathbf{b} vectors, then a search for ω_{opt} may be worthwhile. Table 1.6 presents the results of such a search for the problem considered in Example 1.23. For this problem, $1.05 \leq \omega \leq 1.14$ yields the most efficient solution. Much more dramatic results are obtained for large systems of equations.

1.8. PROGRAMS

Four FORTRAN subroutines for solving systems of linear algebraic equations are presented in this section:

1. Simple Gauss elimination
2. Doolittle LU factorization
3. The Thomas algorithm
4. Successive-over-relaxation (SOR)

The basic computational algorithms are presented as completely self-contained subroutines suitable for use in other programs. Input data and output statements are contained in a main (or driver) program written specifically to illustrate the use of each subroutine.

1.8.1. Simple Gauss Elimination

The elimination step of simple Gauss elimination is based on Eq. (1.100). For each column k ($k = 1, 2, \dots, n - 1$),

$$a_{i,j} = a_{i,j} - (a_{i,k}/a_{k,k})a_{k,j} \quad (i, j = k + 1, k + 2, \dots, n) \quad (1.215a)$$

$$b_i = b_i - (a_{i,k}/a_{k,k})b_k \quad (i = k + 1, k + 2, \dots, n) \quad (1.215b)$$

The back substitution step is based on Eq. (1.101):

$$x_n = b_n/a_{n,n} \quad (1.216a)$$

$$x_i = \frac{b_i - \sum_{j=i+1}^n a_{i,j}x_j}{a_{i,i}} \quad (i = n - 1, n - 2, \dots, 1) \quad (1.216b)$$

A FORTRAN subroutine, *subroutine gauss*, for solving these equations, without pivoting, is presented below. Note that the eliminated elements from matrix A have been replaced by the elimination multipliers, em, so *subroutine gauss* actually evaluates the L and U matrices needed for Doolittle LU factorization, which is presented in Section 1.8.2. *Program main* defines the data set and prints it, calls *subroutine gauss* to implement the solution, and prints the solution.

Program 1.1. Simple Gauss elimination program.

```

program main
c   main program to illustrate linear equation solvers
c   ndim  array dimension, ndim = 6 in this example
c   n      number of equations, n
c   a      coefficient matrix, A(i,j)
c   b      right-hand side vector, b(i)
c   x      solution vector, x(i)
dimension a(6,6),b(6),x(6)
data ndim,n / 6, 3 /
data (a(i,1),i=1,3) / 80.0, -20.0, -20.0 /
data (a(i,2),i=1,3) / -20.0, 40.0, -20.0 /
data (a(i,3),i=1,3) / -20.0, -20.0, 130.0 /
data (b(i),i=1,3) / 20.0, 20.0, 20.0 /
write (6,1000)
do i=1,n
    write (6,1010) i,(a(i,j),j=1,n),b(i)
end do
call gauss (ndim,n,a,b,x)
write (6,1020)
do i=1,n
    write (6,1010) i,(a(i,j),j=1,n),b(i),x(i)
end do
stop
1000 format (' Simple Gauss elimination'// ' ' A and b'/' ')
1010 format (i2,7f12.6)
1020 format (' '/ A, b, and x after elimination'/' ')
end

subroutine gauss (ndim,n,a,b,x)
c   simple gauss elimination
dimension a(ndim,ndim),b(ndim),x(ndim)
c   forward elimination
do k=1,n-1
    do i=k+1,n
        em=a(i,k)/a(k,k)
        a(i,k)=em
        b(i)=b(i)-em*b(k)
        do j=k+1,n
            a(i,j)=a(i,j)-em*a(k,j)
        end do
    end do
end do

```

```

c      back substitution
x(n)=b(n)/a(n,n)
do i=n-1,1,-1
  x(i)=b(i)
  do j=n,i+1,-1
    x(i)=x(i)-a(i,j)*x(j)
  end do
  x(i)=x(i)/a(i,i)
end do
return
end

```

The data set used to illustrate *subroutine gauss* is taken from Example 1.7. The output generated by the simple Gauss elimination program is presented below.

Output 1.1. Solution by simple Gauss elimination.

Simple Gauss elimination

A and b

1	80.000000	-20.000000	-20.000000	20.000000
2	-20.000000	40.000000	-20.000000	20.000000
3	-20.000000	-20.000000	130.000000	20.000000

A, b, and x after elimination

1	80.000000	-20.000000	-20.000000	20.000000	0.600000
2	-0.250000	35.000000	-25.000000	25.000000	1.000000
3	-0.250000	-0.714286	107.142857	42.857143	0.400000

1.8.2. Doolittle LU Factorization

Doolittle LU factorization is based on the LU factorization implicit in Gauss elimination. *Subroutine gauss* presented in Section 1.8.1 is modified to evaluate the **L** and **U** matrices simply by removing the line evaluating $b(i)$ from the first group of statements and entirely deleting the second group of statements, which evaluates the back substitution step. The modified subroutine is named *subroutine lufactor*.

A second subroutine, *subroutine solve*, based on steps 2 and 3 in the description of the Doolittle LU factorization method in Section 1.4, is required to process the **b** vector to the **b'** vector and to process the **b'** vector to the **x** vector. These steps are given by Eqs. (1.139) and (1.140):

$$b'_i = b_i - \sum_{k=1}^{i-1} l_{i,k} b'_k \quad (i = 2, 3, \dots, n) \quad (1.217a)$$

$$x_i = b'_i - \sum_{k=i+1}^n u_{i,k} x_k / u_{i,i} \quad (i = n-1, n-2, \dots, 1) \quad (1.217b)$$

FORTRAN subroutines for implementing Doolittle LU factorization are presented below. *Program main* defines the data set and prints it, calls *subroutine lufactor* to evaluate the **L** and **U** matrices, calls *subroutine solve* to implement the solution for a specified **b**

vector, and prints the solution. *Program main* below shows only the statements which are different from the statements in *program main* in Section 1.8.1.

Program 1.2. Doolittle LU factorization program.

```

program main
c   main program to illustrate linear equation solvers
c   bp   b' vector, bp(i)
dimension a(6,6),b(6),bp(6),x(6)
call lufactor (ndim,n,a)
write (6,1020)
do i=1,n
    write (6,1010) i,(a(i,j),j=1,n)
end do
call solve (ndim,n,a,b,bp,x)
write (6,1030)
do i=1,n
    write (6,1010) i,b(i),bp(i),x(i)
end do
stop
1000 format (' Doolittle LU factorization'// ' ' ' A and b'// ' ')
1010 format (i2,7f12.6)
1020 format (' '// ' L and U stored in A'// ' ')
1030 format (' '// ' b, bprime, and x vectors'// ' ')
end

subroutine lufactor (ndim,n,a)
c   Doolittle LU factorization, stores L and U in A
dimension a(ndim,ndim)
do k=1,n-1
    do i=k+1,n
        em=a(i,k)/a(k,k)
        a(i,k)=em
        do j=k+1,n
            a(i,j)=a(i,j)-em*a(k,j)
        end do
    end do
end do
return
end

subroutine solve (ndim,n,a,b,bp,x)
c   processes b to b' and b' to x
dimension a(ndim,ndim),b(ndim),bp(ndim),x(ndim)
c   forward elimination step to calculate b'
bp(1)=b(1)
do i=2,n
    bp(i)=b(i)
    do j=1,i-1
        bp(i)=bp(i)-a(i,j)*bp(j)
    end do
end do

```

```
c      back substitution step to calculate x
x(n)=bp(n)/a(n,n)
do i=n-1,1,-1
  x(i)=bp(i)
  do j=n,i+1,-1
    x(i)=x(i)-a(i,j)*x(j)
  end do
  x(i)=x(i)/a(i,i)
end do
return
end
```

The data set used to illustrate *subroutines lufactor* and *solve* is taken from Example 1.15. The output generated by the Doolittle LU factorization program is presented below.

Output 1.2. Solution by Doolittle LU factorization.

Doolittle LU factorization

A and b

1	80.000000	-20.000000	-20.000000
2	-20.000000	40.000000	-20.000000
3	-20.000000	-20.000000	130.000000

L and U matrices stored in A matrix

1	80.000000	-20.000000	-20.000000
2	-0.250000	35.000000	-25.000000
3	-0.250000	-0.714286	107.142857

b, bprime, and x vectors

1	20.000000	20.000000	0.600000
2	20.000000	25.000000	1.000000
3	20.000000	42.857143	0.400000

1.8.3. The Thomas Algorithm

The elimination step of the Thomas algorithm is based on Eqs. (1.150) and (1.151):

$$a'_{1,2} = a'_{1,2} \quad (1.218a)$$

$$a'_{i,2} = a'_{i,2} - (a'_{i,1}/a'_{i-1,2})a'_{i-1,3} \quad (i = 2, 3, \dots, n) \quad (1.218b)$$

$$b_1 = b_1 \quad (1.218c)$$

$$b_i = b_i - (a'_{i,1}/a'_{i-1,2})b_{i-1} \quad (i = 2, 3, \dots, n) \quad (1.218d)$$

The back substitution step is based on Eq. (1.152):

$$x_n = b_n/a'_{n,2} \quad (1.219a)$$

$$x_i = (b_i - a'_{i,3}x_{i+1})/a'_{i,2} \quad (i = n-1, n-2, \dots, 1) \quad (1.219b)$$

A FORTRAN subroutine, *subroutine thomas*, for solving these equations is presented below. Note that the eliminated elements from matrix **A** have been replaced by the elimination multipliers, *em*, so *subroutine thomas* actually evaluates the **L** and **U** matrices needed for Doolittle LU factorization. *Program main* defines the data set and prints it, calls *subroutine thomas* to implement the solution, and prints the solution.

Program 1.3. The Thomas algorithm program.

```

program main
c   main program to illustrate linear equation solvers
c   ndim  array dimension, ndim = 9 in this example
c   n      number of equations, n
c   a      coefficient matrix, A(i,j)
c   b      right-hand side vector, b(i)
c   x      solution vector, x(i)
dimension a(9,3),b(9),x(9)
data ndim,n / 9, 7 /
data (a(1,j),j=1,3) /  0.0, -2.25, 1.0 /
data (a(2,j),j=1,3) /  1.0, -2.25, 1.0 /
data (a(3,j),j=1,3) /  1.0, -2.25, 1.0 /
data (a(4,j),j=1,3) /  1.0, -2.25, 1.0 /
data (a(5,j),j=1,3) /  1.0, -2.25, 1.0 /
data (a(6,j),j=1,3) /  1.0, -2.25, 1.0 /
data (a(7,j),j=1,3) /  1.0, -2.25, 0.0 /
data (b(i),i=1,7) / 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -100.0 /
write (6,1000)
do i=1,n
    write (6,1010) i,(a(i,j),j=1,3),b(i)
end do
call thomas (ndim,n,a,b,x)
write (6,1020)
do i=1,n
    write (6,1010) i,(a(i,j),j=1,3),b(i),x(i)
end do
stop
1000 format (' The Thomas algorithm'// ' ' ' A and b'// ' ')
1010 format (i2,6f12.6)
1020 format (' ' ' A, b, and x after elimination'// ' ')
end

subroutine thomas (ndim,n,a,b,x)
c   the Thomas algorithm for a tridiagonal system
dimension a(ndim,3),b(ndim),x(ndim)
c   forward elimination
do i=2,n
    em=a(i,1)/a(i-1,2)
    a(i,1)=em
    a(i,2)=a(i,2)-em*a(i-1,3)
    b(i)=b(i)-a(i,1)*b(i-1)
end do

```

```
c      back substitution
x(n)=b(n)/a(n, 2)
do i=n-1, 1, -1
  x(i)=(b(i)-a(i, 3)*x(i+1))/a(i, 2)
end do
return
end
```

The data set used to illustrate *subroutine thomas* is taken from Example 1.17. The output generated by the Thomas algorithm program is presented below.

Output 1.3. Solution by the Thomas algorithm.

The Thomas algorithm

A and b

1	0.000000	-2.250000	1.000000	0.000000
2	1.000000	-2.250000	1.000000	0.000000
3	1.000000	-2.250000	1.000000	0.000000
4	1.000000	-2.250000	1.000000	0.000000
5	1.000000	-2.250000	1.000000	0.000000
6	1.000000	-2.250000	1.000000	0.000000
7	1.000000	-2.250000	0.000000	-100.000000

A, b, and x after elimination

1	0.000000	-2.250000	1.000000	0.000000	1.966751
2	-0.444444	-1.805556	1.000000	0.000000	4.425190
3	-0.553846	-1.696154	1.000000	0.000000	7.989926
4	-0.589569	-1.660431	1.000000	0.000000	13.552144
5	-0.602253	-1.647747	1.000000	0.000000	22.502398
6	-0.606889	-1.643111	1.000000	0.000000	37.078251
7	-0.608602	-1.641398	0.000000	-100.000000	60.923667

1.8.4. Successive-Over-Relaxation (SOR)

Successive-over-relaxation (SOR) is based on Eqs. (1.212) and (1.213):

$$x_i^{(k+1)} = x_i^{(k)} + \omega \frac{R_i^{(k)}}{a_{i,i}} \quad (i = 1, 2, \dots, n) \quad (1.220a)$$

$$R_i^{(k)} = b_i - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i}^n a_{i,j} x_j^{(k)} \quad (i = 1, 2, \dots, n) \quad (1.220b)$$

A FORTRAN subroutine, *subroutine sor*, for solving these equations is presented below. *Program main* defines the data set and prints it, calls *subroutine sor* to implement the solution, and prints the solution. Input variable *iw* is a flag for output of intermediate results. When *iw* = 0, no intermediate results are output. When *iw* = 1, intermediate results are output.

Program 1.4. Successive-over-relaxation (SOR) program.

```

program main
c   main program to illustrate linear equation solvers
c   ndim  array dimension, ndim = 6 in this example
c   n      number of equations, n
c   a      coefficient matrix, A(i,j)
c   b      right-hand side vector, b(i)
c   x      solution vector, x(i)
c   iter  number of iterations allowed
c   tol   convergence tolerance
c   omega over-relaxation factor
c   iw    flag for intermediate output: 0 no, 1 yes
dimension a(6,6),b(6),x(6)
data ndim,n,iter,tol,omega,iw / 6,5,25,0.000001,1.0,1 /
data (a(i,1),i=1,5) / 4.0, -1.0,  0.0,  1.0,  0.0 /
data (a(i,2),i=1,5) / -1.0,  4.0, -1.0,  0.0,  1.0 /
data (a(i,3),i=1,5) /  0.0, -1.0,  4.0, -1.0,  0.0 /
data (a(i,4),i=1,5) /  1.0,  0.0, -1.0,  4.0, -1.0 /
data (a(i,5),i=1,5) /  0.0,  1.0,  0.0, -1.0,  4.0 /
data (b(i),i=1,5) / 100.0, 100.0, 100.0, 100.0, 100.0 /
data (x(i),i=1,5) / 0.0, 0.0, 0.0, 0.0, 0.0 /
write (6,1000)
do i=1,n
    write (6,1010) i,(a(i,j),j=1,n),b(i)
end do
write (6,1020)
it=0
write (6,1010) it,(x(i),i=1,n)
call sor (ndim,n,a,b,x,iter,tol,omega,iw,it)
if (iw.eq.0) write (6,1010) it,(x(i),i=1,n)
stop
1000 format (' SOR iteration'// ' A and b'// ' ')
1010 format (i2,7f12.6)
1020 format (' '// i           x(1) to x(n)'// ' ')
end

subroutine sor (ndim,n,a,b,x,iter,tol,omega,iw,it)
c   sor iteration
dimension a(ndim,ndim),b(ndim),x(ndim)
do it=1,iter
    dxmax=0.0
    do i=1,n
        residual=b(i)
        do j=1,n
            residual=residual-a(i,j)*x(j)
        end do
        if (abs(residual).gt.dxmax) dxmax=abs(residual)
        x(i)=x(i)+omega*residual/a(i,i)
    end do
end

```

```

    if (iw.eq.1) write (6,1000) it,(x(i),i=1,n)
    if (dxmax.lt.tol) return
end do
write (6,1010)
return
1000 format (i2,7f12.6)
1010 format (' '' Solution failed to converge'' ')
end

```

The data set used to illustrate *subroutine sor* is taken from Example 1.23. The output generated by the SOR program is presented below.

Output 1.4. Solution by successive-over-relaxation (SOR).

SOR iteration

A and b

1	4.000000	-1.000000	0.000000	1.000000	0.000000	100.000000
2	-1.000000	4.000000	-1.000000	0.000000	1.000000	100.000000
3	0.000000	-1.000000	4.000000	-1.000000	0.000000	100.000000
4	1.000000	0.000000	-1.000000	4.000000	-1.000000	100.000000
5	0.000000	1.000000	0.000000	-1.000000	4.000000	100.000000

i *x(1) to x(n)*

0	0.000000	0.000000	0.000000	0.000000	0.000000
1	25.000000	31.250000	32.812500	26.953125	23.925781
2	26.074219	33.740234	40.173340	34.506226	25.191498
3	24.808502	34.947586	42.363453	35.686612	25.184757
4	24.815243	35.498485	42.796274	35.791447	25.073240
.....
15	25.000000	35.714286	42.857143	35.714286	25.000000
16	25.000000	35.714286	42.857143	35.714286	25.000000

1.8.5. Packages for Systems of Linear Algebraic Equations

Numerous libraries and software packages are available for solving systems of linear algebraic equations. Many work stations and mainframe computers have such libraries attached to their operating systems. If not, libraries such as IMSL (International Mathematics and Statistics Library) or LINPACK (Argonne National Laboratory) can be added to the operating systems.

Most commercial software packages contain solvers for systems of linear algebraic equations. Some of the more prominent packages are Matlab and Mathcad. The spreadsheet Excel can also be used to solve systems of equations. More sophisticated packages, such as Mathematica, Macsyma, and Maple, also contain linear equation solvers. Finally, the book *Numerical Recipes* (Press et al., 1989) contains several subroutines for solving systems of linear algebraic equations.

1.9 SUMMARY

The basic methods for solving systems of linear algebraic equations are presented in this chapter. Some general guidelines for selecting a method for solving systems of linear algebraic equations are given below.

- Direct elimination methods are preferred for small systems ($n \lesssim 50$ to 100) and systems with few zeros (nonsparse systems). Gauss elimination is the method of choice.
- For tridiagonal systems, the Thomas algorithm is the method of choice.
- LU factorization methods (e.g., the Doolittle method) are the methods of choice when more than one \mathbf{b} vector must be considered.
- For large systems that are not diagonally dominant, the round-off errors can be large.
- Iterative methods are preferred for large, sparse matrices that are diagonally dominant. The SOR method is the method of choice. Numerical experimentation to find the optimum over-relaxation factor ω_{opt} is usually worthwhile if the system of equations is to be solved for many \mathbf{b} vectors.

After studying Chapter 1, you should be able to:

1. Describe the general structure of a system of linear algebraic equations.
2. Explain the solution possibilities for a system of linear algebraic equations:
 - (a) a unique solution, (b) no solution, (c) an infinite number of solutions, and (d) the trivial solution.
3. Understand the differences between direct elimination methods and iterative methods.
4. Understand the elementary properties of matrices and determinants.
5. Recognize the structures of square matrices, diagonal matrices, the identity matrix, upper and lower triangular matrices, tridiagonal matrices, banded matrices, and singular matrices.
6. Perform elementary matrix algebra.
7. Understand the concept of diagonal dominance.
8. Understand the concept of the inverse of a matrix.
9. Define the determinant of a matrix.
10. Express a system of linear algebraic equations in matrix form.
11. Understand matrix row operations.
12. Explain the general concept behind direct elimination methods.
13. Apply Cramer's rule to solve a system of linear algebraic equations.
14. Solve a system of linear algebraic equations by Gauss elimination.
15. Understand pivoting and scaling in Gauss elimination.
16. Evaluate 2×2 and 3×3 determinants by the diagonal method.
17. Evaluate a determinant by the cofactor method.
18. Evaluate a determinant by Gauss elimination.
19. Solve a system of linear algebraic equations by Gauss-Jordan elimination.
20. Determine the inverse of a matrix by Gauss-Jordan elimination.
21. Solve a system of linear algebraic equations by the matrix inverse method.
22. Explain the concept of matrix factorization.
23. Explain how Doolittle LU factorization is obtained by Gauss elimination.
24. Solve a system of linear algebraic equations by Doolittle LU factorization.
25. Determine the inverse of a matrix by Doolittle LU factorization.

26. Understand the special properties of a tridiagonal matrix.
27. Solve a tridiagonal system of linear algebraic equation by the Thomas algorithm.
28. Understand the concept of block tridiagonal systems of linear algebraic equations.
29. Understand the pitfalls of direct elimination methods.
30. Explain the effects of round-off on numerical algorithms.
31. Explain the concept of system condition.
32. Describe the effects of ill-conditioning.
33. Define the norm of a vector or matrix.
34. Define the condition number of a matrix.
35. Explain the significance of the condition number of a matrix.
36. Explain the general concept behind iterative methods.
37. Understand the structure and significance of a sparse matrix.
38. Explain the importance of diagonal dominance for iterative methods.
39. Solve a system of linear algebraic equations by Jacobi iteration.
40. Solve a system of linear algebraic equations by Gauss-Seidel iteration.
41. Solve a system of linear algebraic equations by successive-over relaxation (SOR).
42. Appreciate the importance of using the optimum overrelaxation factor, ω_{opt} .
43. Define the meaning and significance of the residual in an iterative method.
44. Explain accuracy of an approximate method.
45. Understand the difference between absolute error and relative error.
46. Understand convergence of an iterative method and convergence criteria.
47. Choose a method for solving a system of linear algebraic equations based on its size and structure.

EXERCISE PROBLEMS

Section 1.2. Properties of Matrices and Determinants

The following four matrices are considered in this section:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 3 \\ 5 & 3 & 1 \\ 2 & 3 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 2 & 3 & 5 \\ 3 & 1 & -2 \\ 1 & 3 & 4 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 2 & 1 \\ 3 & 4 \\ 2 & 5 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 3 \\ 1 & 3 & 5 \end{bmatrix}$$

1. Determine the following quantities, if defined: (a) $\mathbf{A} + \mathbf{B}$, (b) $\mathbf{B} + \mathbf{A}$, (c) $\mathbf{A} + \mathbf{D}$, (d) $\mathbf{A} + \mathbf{C}$, (e) $\mathbf{B} + \mathbf{C}$.
2. Determine the following quantities, if defined: (a) $\mathbf{A} - \mathbf{B}$, (b) $\mathbf{B} - \mathbf{A}$, (c) $\mathbf{A} - \mathbf{D}$, (d) $\mathbf{B} - \mathbf{D}$, (e) $\mathbf{A} - \mathbf{C}$, (f) $\mathbf{B} - \mathbf{C}$, (g) $\mathbf{C} - \mathbf{B}$, (h) $\mathbf{D} - \mathbf{C}$.
3. Determine the following quantities, if defined: (a) \mathbf{AC} , (b) \mathbf{BC} , (c) \mathbf{CA} , (d) \mathbf{CB} , (e) \mathbf{AD} , (f) \mathbf{BD} , (g) $\mathbf{C}^T \mathbf{A}$, (h) $\mathbf{C}^T \mathbf{B}$, (i) $\mathbf{C}^T \mathbf{D}$, (j) \mathbf{AA}^T , (k) \mathbf{BB}^T , (l) \mathbf{DD}^T .
4. (a) Compute \mathbf{AB} and \mathbf{BA} and show that $\mathbf{AB} \neq \mathbf{BA}$. (b) Compute \mathbf{AD} and \mathbf{DA} and show that $\mathbf{AD} \neq \mathbf{DA}$. (c) Compute \mathbf{BD} and \mathbf{DB} and show that $\mathbf{BD} \neq \mathbf{DB}$.
5. Show that: (a) $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$ and (b) $\mathbf{AB} = (\mathbf{Ab}_1 \quad \mathbf{Ab}_2 \quad \mathbf{Ab}_3)$, where \mathbf{b}_1 , \mathbf{b}_2 , and \mathbf{b}_3 are the columns of \mathbf{B} .
6. Work Problem 5 for the general 3×3 matrices \mathbf{A} and \mathbf{B} .
7. Verify that (a) $\mathbf{A} + (\mathbf{B} + \mathbf{D}) = (\mathbf{A} + \mathbf{B}) + \mathbf{D}$ and (b) $\mathbf{A}(\mathbf{BD}) = (\mathbf{AB})\mathbf{D}$.

8. Calculate the following determinants by the diagonal method, if defined:
 - (a) $\det(\mathbf{A})$
 - (b) $\det(\mathbf{B})$
 - (c) $\det(\mathbf{C})$
 - (d) $\det(\mathbf{D})$
 - (e) $\det(\mathbf{AB})$
 - (f) $\det(\mathbf{AD})$
 - (g) $\det(\mathbf{BA})$
 - (h) $\det(\mathbf{DA})$
 - (i) $\det(\mathbf{CD})$
 - (j) $\det(\mathbf{C}^T \mathbf{A})$
9. Work Problem 8 using the cofactor method.
10. Show that $\det(\mathbf{A}) \det(\mathbf{B}) = \det(\mathbf{AB})$.
11. Show that $\det(\mathbf{A}) \det(\mathbf{D}) = \det(\mathbf{AD})$.
12. Show that for the general 2×2 matrices \mathbf{A} and \mathbf{B} , $\det(\mathbf{A}) \det(\mathbf{B}) = \det(\mathbf{AB})$.

Section 1.3. Direct Elimination Methods

Consider the following eight systems of linear algebraic equations, $\mathbf{Ax} = \mathbf{b}$:

$$\begin{array}{l} -2x_1 + 3x_2 + x_3 = 9 \\ 3x_1 + 4x_2 - 5x_3 = 0 \\ x_1 - 2x_2 + x_3 = -4 \end{array} \quad (\text{A}) \quad \left[\begin{array}{ccc|c} 1 & 1 & 3 & x \\ 5 & 3 & 1 & y \\ 2 & 3 & 1 & z \end{array} \right] = \left[\begin{array}{c} 2 \\ 3 \\ -1 \end{array} \right] \quad (\text{B})$$

$$\begin{array}{l} x_1 + 3x_2 + 2x_3 - x_4 = 9 \\ 4x_1 + 2x_2 + 5x_3 + x_4 = 27 \\ 3x_1 - 3x_2 + 2x_3 + 4x_4 = 19 \\ -x_1 + 2x_2 - 3x_3 + 5x_4 = 14 \end{array} \quad (\text{C}) \quad \left[\begin{array}{cccc|c} 3 & 1 & -1 & 3 & 4 \\ 2 & 1 & -2 & 0 & -1 \\ 0 & 3 & 2 & -2 & 4 \\ 1 & 1 & 1 & 5 & -2 \end{array} \right] [x_i] = \left[\begin{array}{c} 4 \\ -1 \\ 4 \\ -2 \end{array} \right] \quad (\text{D})$$

$$\left[\begin{array}{ccc|c} 1 & -2 & 1 & x_1 \\ 2 & 1 & 2 & x_2 \\ -1 & 1 & 3 & x_3 \end{array} \right] = \left[\begin{array}{c} -1 \\ 3 \\ 8 \end{array} \right] \quad (\text{E}) \quad \left[\begin{array}{ccc|c} 2 & 3 & 5 & x_1 \\ 3 & 1 & -2 & x_2 \\ 1 & 3 & 4 & x_3 \end{array} \right] = \left[\begin{array}{c} 0 \\ -2 \\ -3 \end{array} \right] \quad (\text{F})$$

$$\left[\begin{array}{cccc|c} 2 & -2 & 2 & 1 & x_1 \\ 2 & -4 & 1 & 3 & x_2 \\ -1 & 3 & -4 & 2 & x_3 \\ 2 & 4 & 3 & -2 & x_4 \end{array} \right] = \left[\begin{array}{c} 7 \\ 10 \\ -14 \\ 1 \end{array} \right] \quad (\text{G}) \quad \left[\begin{array}{ccc|c} 1 & 1 & 1 & x_1 \\ 1 & 2 & 1 & x_2 \\ 3 & 3 & 4 & x_3 \end{array} \right] = \left[\begin{array}{c} 0 \\ -4 \\ 1 \end{array} \right] \quad (\text{H})$$

Cramer's Rule

13. Solve Eq. (A) by Cramer's rule.
14. Solve Eq. (B) by Cramer's rule.
15. Solve Eq. (C) by Cramer's rule.
16. Solve Eq. (D) by Cramer's rule.
17. Solve Eq. (E) by Cramer's rule.
18. Solve Eq. (F) by Cramer's rule.
19. Solve Eq. (G) by Cramer's rule.
20. Solve Eq. (H) by Cramer's rule.

Gauss Elimination

21. Solve Eq. (A) by Gauss elimination without pivoting.
22. Solve Eq. (B) by Gauss elimination without pivoting.
23. Solve Eq. (C) by Gauss elimination without pivoting.
24. Solve Eq. (D) by Gauss elimination without pivoting.
25. Solve Eq. (E) by Gauss elimination without pivoting.
26. Solve Eq. (F) by Gauss elimination without pivoting.
27. Solve Eq. (G) by Gauss elimination without pivoting.
28. Solve Eq. (H) by Gauss elimination without pivoting.

Gauss-Jordan Elimination

29. Solve Eq. (A) by Gauss-Jordan elimination.
30. Solve Eq. (B) by Gauss-Jordan elimination.
31. Solve Eq. (C) by Gauss-Jordan elimination.
32. Solve Eq. (D) by Gauss-Jordan elimination.
33. Solve Eq. (E) by Gauss-Jordan elimination.
34. Solve Eq. (F) by Gauss-Jordan elimination.
35. Solve Eq. (G) by Gauss-Jordan elimination.
36. Solve Eq. (H) by Gauss-Jordan elimination.

The Matrix Inverse Method

37. Solve Eq. (A) using the matrix inverse method.
38. Solve Eq. (B) using the matrix inverse method.
39. Solve Eq. (C) using the matrix inverse method.
40. Solve Eq. (D) using the matrix inverse method.
41. Solve Eq. (E) using the matrix inverse method.
42. Solve Eq. (F) using the matrix inverse method.
43. Solve Eq. (G) using the matrix inverse method.
44. Solve Eq. (H) using the matrix inverse method.

Section 1.4. LU Factorization

45. Solve Eq. (A) by the Doolittle LU factorization method.
46. Solve Eq. (B) by the Doolittle LU factorization method.
47. Solve Eq. (C) by the Doolittle LU factorization method.
48. Solve Eq. (D) by the Doolittle LU factorization method.
49. Solve Eq. (E) by the Doolittle LU factorization method.
50. Solve Eq. (F) by the Doolittle LU factorization method.
51. Solve Eq. (G) by the Doolittle LU factorization method.
52. Solve Eq. (H) by the Doolittle LU factorization method.

Section 1.5. Tridiagonal Systems of Equations

Consider the following tridiagonal systems of linear algebraic equations:

$$\begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 12 \\ 11 \end{bmatrix} \quad (\text{I}) \quad \begin{bmatrix} 3 & 2 & 0 & 0 \\ 2 & 3 & 2 & 0 \\ 0 & 2 & 3 & 2 \\ 0 & 0 & 2 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 17 \\ 14 \\ 7 \end{bmatrix} \quad (\text{J})$$

$$\begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ -7 \\ -1 \end{bmatrix} \quad (\text{K}) \quad \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 0 \\ 8 \end{bmatrix} \quad (\text{L})$$

$$\begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix} [x_i] = \begin{bmatrix} 150 \\ 200 \\ 150 \\ 100 \end{bmatrix} \quad (\text{M}) \quad \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 0 \\ 8 \end{bmatrix} \quad (\text{N})$$

53. Solve Eq. (I) by the Thomas algorithm.
54. Solve Eq. (J) by the Thomas algorithm.
55. Solve Eq. (K) by the Thomas algorithm.
56. Solve Eq. (L) by the Thomas algorithm.
57. Solve Eq. (M) by the Thomas algorithm.
58. Solve Eq. (N) by the Thomas algorithm.

Section 1.7. Iterative Methods

Solve the following problems by iterative methods. Let $\mathbf{x}^{(0)T} = [0.0 \quad 0.0 \quad 0.0 \quad 0.0]$. For hand calculations, make at least five iterations. For computer solutions, iterate until six digits after the decimal place converges.

Jacobi Iteration

59. Solve Eq. (I) by Jacobi iteration.
60. Solve Eq. (K) by Jacobi iteration.
61. Solve Eq. (L) by Jacobi iteration.
62. Solve Eq. (M) by Jacobi iteration.
63. Solve Eq. (N) by Jacobi iteration.

Gauss-Seidel Iteration

64. Solve Eq. (I) by Gauss-Seidel iteration.
65. Solve Eq. (K) by Gauss-Seidel iteration.
66. Solve Eq. (L) by Gauss-Seidel iteration.
67. Solve Eq. (M) by Gauss-Seidel iteration.
68. Solve Eq. (N) by Gauss-Seidel iteration.

Successive Over-Relaxation

69. Solve Eq. (I) by the SOR method with $\omega = 1.27$.
70. Solve Eq. (K) by the SOR method with $\omega = 1.27$.
71. Solve Eq. (L) by the SOR method with $\omega = 1.27$.
72. Solve Eq. (M) by the SOR method with $\omega = 1.05$.
73. Solve Eq. (N) by the SOR method with $\omega = 1.25$.
74. Solve Eq. (I) by the SOR method for $1.25 \leq \omega \leq 1.35$ with $\Delta\omega = 0.01$.
75. Solve Eq. (K) by the SOR method for $1.25 \leq \omega \leq 1.35$ with $\Delta\omega = 0.01$.
76. Solve Eq. (L) by the SOR method for $1.25 \leq \omega \leq 1.35$ with $\Delta\omega = 0.01$.
77. Solve Eq. (M) by the SOR method for $1.00 \leq \omega = 1.10$ with $\Delta\omega = 0.01$.
78. Solve Eq. (N) by the SOR method for $1.25 \leq \omega = 1.35$ with $\Delta\omega = 0.01$.

Section 1.8. Programs

79. Implement the simple Gauss elimination program presented in Section 1.8.1. Check out the program using the given data.
80. Solve any of Eqs. (A) to (H) using the Gauss elimination program.
81. Implement the Doolittle LU factorization program presented in Section 1.8.2. Check out the program using the given data.
82. Solve any of Eqs. (A) to (H) using the Doolittle LU factorization program.
83. Implement the Thomas algorithm program presented in Section 1.8.3. Check out the program using the given data.
84. Solve any of Eqs. (I) to (N) using the Thomas algorithm program.
85. Implement the SOR program presented in Section 1.8.4. Check out the program using the given data.
86. Solve any of Eqs. (I) and (K) to (N) using the SOR program.

2

Eigenproblems

- 2.1. Introduction
- 2.2. Mathematical Characteristics of Eigenproblems
- 2.3. The Power Method
- 2.4. The Direct Method
- 2.5. The QR Method
- 2.6. Eigenvectors
- 2.7. Other Methods
- 2.8. Programs
- 2.9. Summary
Problems

Examples

- 2.1. The direct power method
- 2.2. The inverse power method
- 2.3. The shifted direct power method for opposite extreme eigenvalues
- 2.4. The shifted inverse power method for intermediate eigenvalues
- 2.5. The shifted inverse power method for accelerating convergence
- 2.6. The direct method for a linear eigenproblem
- 2.7. The direct method for a nonlinear eigenproblem
- 2.8. The basic QR method
- 2.9. Eigenvectors

2.1 INTRODUCTION

Consider the dynamic mechanical spring-mass system illustrated in Figure 2.1. Applying Newton's second law of motion, $\sum F = m\ddot{x}$, to each individual mass gives

$$K_2(x_2 - x_1) + K_3(x_3 - x_1) - K_1x_1 = m_1\ddot{x}_1 \quad (2.1a)$$

$$-K_2(x_2 - x_1) + K_4(x_3 - x_2) = m_2\ddot{x}_2 \quad (2.1b)$$

$$-K_3(x_3 - x_1) - K_4(x_3 - x_2) - K_5x_3 = m_3\ddot{x}_3 \quad (2.1c)$$

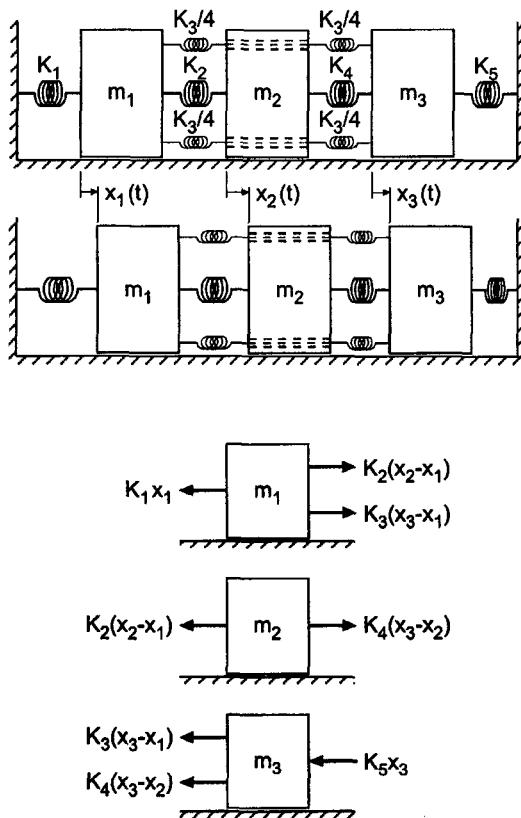


Figure 2.1 Dynamic mechanical spring-mass system.

Rearranging Eq. (2.1) yields:

$$-(K_1 + K_2 + K_3)x_1 + K_2x_2 + K_3x_3 = m_1\ddot{x}_1 \quad (2.2a)$$

$$K_2x_1 - (K_2 + K_4)x_2 + K_4x_3 = m_2\ddot{x}_2 \quad (2.2b)$$

$$K_3x_1 + K_4x_2 - (K_3 + K_4 + K_5)x_3 = m_3\ddot{x}_3 \quad (2.2c)$$

For steady periodic motion at large time,

$$\mathbf{x}(t) = \mathbf{X} \sin(\omega t) \quad (2.3a)$$

where $\mathbf{x}(t)^T = [x_1(t) \ x_2(t) \ x_3(t)]$, $\mathbf{X}^T = [X_1 \ X_2 \ X_3]$ is the amplitude of the oscillation of the masses, and ω is the undamped natural frequency of the system. Differentiating Eq. (2.3a) gives

$$\frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}} = \omega\mathbf{X} \cos(\omega t) \quad \text{and} \quad \frac{d^2\mathbf{x}}{dt^2} = \ddot{\mathbf{x}} = -\omega^2\mathbf{X} \sin(\omega t) \quad (2.3b)$$

Substituting Eq. (2.3b) into Eq. (2.2) gives

$$-(K_1 + K_2 + K_3)X_1 + K_2X_2 + K_3X_3 = -m_1\omega^2X_1 \quad (2.4a)$$

$$K_2X_1 - (K_2 + K_4)X_2 + K_4X_3 = -m_2\omega^2X_2 \quad (2.4b)$$

$$K_3X_1 + K_4X_2 - (K_3 + K_4 + K_5)X_3 = -m_3\omega^2X_3 \quad (2.4c)$$

Rearranging Eq. (2.4) yields the *system equation*:

$$(K_1 + K_2 + K_3 - m_1 \omega^2)X_1 - K_2 X_2 - K_3 X_3 = 0 \quad (2.5a)$$

$$-K_2 X_1 + (K_2 + K_4 - m_2 \omega^2)X_2 - K_4 X_3 = 0 \quad (2.5b)$$

$$-K_3 X_1 - K_4 X_2 + (K_3 + K_4 + K_5 - m_3 \omega^2)X_3 = 0 \quad (2.5c)$$

Let's nondimensionalize Eq. (2.5) using K_{ref} and m_{ref} . Thus, $\bar{m} = m/m_{\text{ref}}$ and $\bar{K} = K/K_{\text{ref}}$. Substituting these definitions into Eq. (2.5) and dividing by K_{ref} gives:

$$\left[\bar{K}_1 + \bar{K}_2 + \bar{K}_3 - \bar{m}_1 \left(\frac{m_{\text{ref}} \omega^2}{K_{\text{ref}}} \right) \right] X_1 - \bar{K}_2 X_2 - \bar{K}_3 X_3 = 0 \quad (2.6a)$$

$$-\bar{K}_2 X_1 + \left[\bar{K}_2 + \bar{K}_4 - \bar{m}_2 \left(\frac{m_{\text{ref}} \omega^2}{K_{\text{ref}}} \right) \right] X_2 - \bar{K}_4 X_3 = 0 \quad (2.6b)$$

$$-\bar{K}_3 X_1 - \bar{K}_4 X_2 + \left[\bar{K}_3 + \bar{K}_4 + \bar{K}_5 - \bar{m}_3 \left(\frac{m_{\text{ref}} \omega^2}{K_{\text{ref}}} \right) \right] X_3 = 0 \quad (2.6c)$$

Define the parameter λ as follows:

$$\boxed{\lambda = \frac{m_{\text{ref}} \omega^2}{K_{\text{ref}}}} \quad (2.7)$$

Substituting Eq. (2.7) into Eq. (2.6) gives the *nondimensional system equation*:

$$(\bar{K}_1 + \bar{K}_2 + \bar{K}_3 - \bar{m}_1 \lambda)X_1 - \bar{K}_2 X_2 - \bar{K}_3 X_3 = 0 \quad (2.8a)$$

$$-\bar{K}_2 X_1 + (\bar{K}_2 + \bar{K}_4 - \bar{m}_2 \lambda)X_2 - \bar{K}_4 X_3 = 0 \quad (2.8b)$$

$$-\bar{K}_3 X_1 - \bar{K}_4 X_2 + (\bar{K}_3 + \bar{K}_4 + \bar{K}_5 - \bar{m}_3 \lambda)X_3 = 0 \quad (2.8c)$$

Consider a specific system for which $K_1 = 40 \text{ N/cm}$, $K_2 = K_3 = K_4 = 20 \text{ N/cm}$, and $K_5 = 90 \text{ N/cm}$, and $m_1 = m_2 = m_3 = 2 \text{ kg}$. Let $K_{\text{ref}} = 10 \text{ N/cm}$ and $m_{\text{ref}} = 2 \text{ kg}$. For these values, Eq. (2.8) becomes:

$$\boxed{(8 - \lambda)X_1 - 2X_2 - 2X_3 = 0} \quad (2.9a)$$

$$\boxed{-2X_1 + (4 - \lambda)X_2 - 2X_3 = 0} \quad (2.9b)$$

$$\boxed{-2X_1 - 2X_2 + (13 - \lambda)X_3 = 0} \quad (2.9c)$$

Equation (2.9) is a system of three homogeneous linear algebraic equations. There are four unknowns: X_1 , X_2 , X_3 , and λ (i.e., ω). Clearly unique values of the four unknowns cannot be determined by three equations. In fact, the only solution, other than the trivial solution $\mathbf{X} = 0$, depends on the special values of λ , called *eigenvalues*. Equation (2.9) is a classical *eigenproblem*. The values of λ that satisfy Eq. (2.9) are called *eigenvalues*. Unique values of $\mathbf{X}^T = [X_1 \ X_2 \ X_3]$ cannot be determined. However, for every value of λ , relative values of X_1 , X_2 , and X_3 can be determined. The corresponding values of \mathbf{X} are called *eigenvectors*. The eigenvectors determine the *mode* of oscillation (i.e., the relative values of X_1 , X_2 , X_3). Equation (2.9) can be written as

$$\boxed{(\mathbf{A} - \lambda \mathbf{I})\mathbf{X} = 0} \quad (2.10)$$

where

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \quad (2.11)$$

Equation (2.10) is a system of homogeneous linear algebraic equations. Equation (2.10) is the classical form of an eigenproblem.

Chapter 1 is devoted to the solution of systems of nonhomogeneous linear algebraic equations:

$$\mathbf{Ax} = \mathbf{b} \quad (2.12)$$

As discussed in Section 1.1, Eq. (2.12) may have a unique solution (the case considered in Chapter 1), no solution, an infinite number of solutions, or the trivial solution, $\mathbf{x} = 0$, if the system of equations is homogeneous:

$$\mathbf{Ax} = \mathbf{0} \quad (2.13)$$

Chapter 2 is concerned with solutions to Eq. (2.13) other than $\mathbf{x} = 0$, which are possible if the coefficient matrix \mathbf{A} contains an unknown parameter, called an *eigenvalue*.

Every nonsingular square $n \times n$ matrix \mathbf{A} has a set of n eigenvalues λ_i ($i = 1, \dots, n$) and n eigenvectors \mathbf{x}_i ($i = 1, \dots, n$) that satisfy the equation

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{x} = 0 \quad \text{or} \quad \mathbf{Ax} = \lambda \mathbf{x} \quad (2.14)$$

The eigenvalues may be real or complex and distinct or repeated. The elements of the corresponding eigenvectors \mathbf{x}_i are not unique. However, their relative magnitudes can be determined.

Consider an eigenproblem specified by two homogeneous linear algebraic equations:

$$(a_{11} - \lambda)x_1 + a_{12}x_2 = 0 \quad (2.15a)$$

$$(a_{21}x_1 + (a_{22} - \lambda)x_2 = 0 \quad (2.15b)$$

Equation (2.15) represents two straight lines in the x_1x_2 plane, both passing through the origin $x_1 = x_2 = 0$. Rearranging Eq. (2.15) gives

$$x_2 = -\frac{(a_{11} - \lambda)}{a_{12}}x_1 = m_1x_1 \quad (2.16a)$$

$$x_2 = -\frac{a_{21}}{(a_{22} - \lambda)}x_1 = m_2x_1 \quad (2.16b)$$

where m_1 and m_2 are the slopes of the two straight lines. Figure 2.2 illustrates Eq. (2.16) in the x_1x_2 plane. Both straight lines pass through the origin where $x_1 = x_2 = 0$, which is the trivial solution. If the slopes m_1 and m_2 are different, there is no other solution, as illustrated in Figure 2.2a. However, if $m_1 = m_2 = m$, as illustrated in Figure 2.2b, then the two straight lines lie on top of each other, and there are an infinite number of solutions. For any value of x_1 , there is a corresponding value of x_2 . The ratio of x_2 to x_1 is specified by value of the slope m . The values of λ which make $m_1 = m_2 = m$ are called *eigenvalues*, and the solution vector \mathbf{x} corresponding to λ is called an *eigenvector*. Problems involving eigenvalues and eigenvectors are called *eigenproblems*.

Eigenproblems arise in the analysis of many physical systems. They arise in the analysis of the dynamic behavior of mechanical, electrical, fluid, thermal, and structural

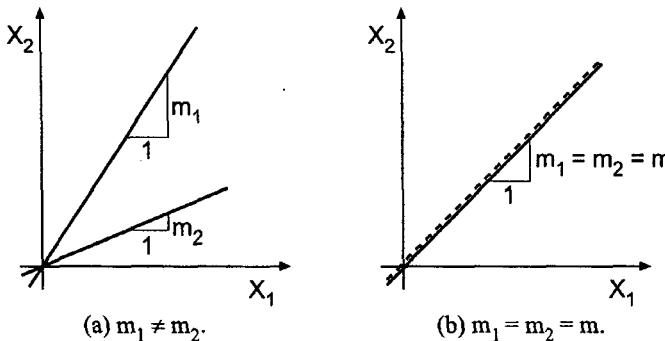


Figure 2.2 Graphical representation of Eq. (2.16).

systems. They also arise in the analysis of control systems. The objectives of this chapter are to introduce the general features of eigenproblems, to present several methods for solving simple eigenproblems, and to illustrate those methods by examples.

There are several methods for solving eigenproblems. Equation (2.14) can be solved *directly* by setting the determinant of $(\mathbf{A} - \lambda \mathbf{I})$ equal to zero and solving the resulting polynomial, which is called the *characteristic equation*, for λ . An iterative method, called the *power method*, is based on the repetitive matrix multiplication of an assumed eigenvector \mathbf{x} by matrix \mathbf{A} , which eventually yields both λ and \mathbf{x} . The power method and the direct method are illustrated in this chapter. A more general and more powerful method, called the QR method, is based on more advanced concepts. The QR method is presented in Section 2.5. Serious students of eigenproblems should use the QR method.

The organization of Chapter 2 is illustrated in Figure 2.3. After a discussion of the general features of eigenproblems in this section, the mathematical characteristics of eigenproblems are discussed in Section 2.2. The power method and its variations are presented in Section 2.3. Section 2.4 presents the direct method. The most powerful method, the QR method, is developed in Section 2.5. The evaluation of eigenvectors is discussed in Section 2.6. A brief mention of other methods is then presented. Two programs for solving eigenproblems follow. The chapter closes with a Summary, which presents some general philosophy about solving eigenproblems, and lists the things you should be able to do after studying Chapter 2.

2.2 MATHEMATICAL CHARACTERISTICS OF EIGENPROBLEMS

The general features of eigenproblems are introduced in Section 2.1. The mathematical characteristics of eigenproblems are presented in this section.

Consider a system of nonhomogeneous linear algebraic equations:

$$\mathbf{C}\mathbf{x} = \mathbf{b} \quad (2.17)$$

Solving for \mathbf{x} by Cramer's rule yields

$$x_j = \frac{\det(\mathbf{C}^j)}{\det(\mathbf{C})} \quad (j = 1, \dots, n) \quad (2.18)$$

where matrix \mathbf{C}^j is matrix \mathbf{C} with column j replaced by the vector \mathbf{b} . In general $\det(\mathbf{C}) \neq 0$, and unique values are found for x_j .

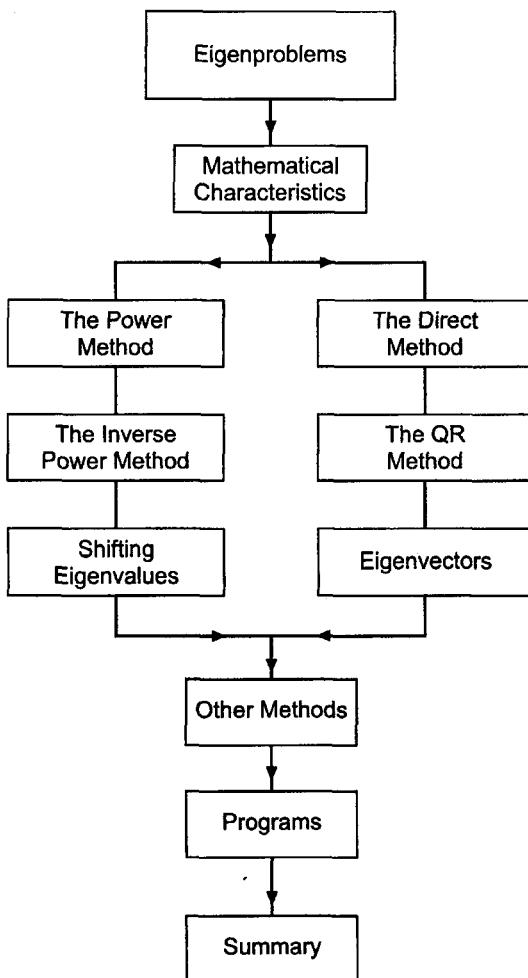


Figure 2.3 Organization of Chapter 2.

Consider a system of homogeneous linear algebraic equations:

$$\mathbf{C}\mathbf{x} = \mathbf{0} \quad (2.19)$$

Solving for \mathbf{x} by Cramer's rule yields

$$x_j = \frac{\det(\mathbf{C}'^j)}{\det(\mathbf{C})} = \frac{0}{|\mathbf{C}|} \quad (j = 1, \dots, n) \quad (2.20)$$

Therefore, $\mathbf{x} = \mathbf{0}$ unless $\det(\mathbf{C}) = 0$. In general, $\det(\mathbf{C}) \neq 0$, and the only solution is the trivial solution, $\mathbf{x} = \mathbf{0}$. For certain forms of \mathbf{C} that involve an unspecified arbitrary scalar λ , the value of λ can be chosen to force $\det(\mathbf{C}) = 0$, so that a solution other than the trivial solution, $\mathbf{x} = \mathbf{0}$, is possible. In that case \mathbf{x} is not unique, but relative values of x_j can be found.

Consider the coefficient matrix \mathbf{C} to be of the form

$$\mathbf{C} = \mathbf{A} - \lambda\mathbf{B} \quad (2.21)$$

where λ is an unspecified scalar. Then

$$\mathbf{C}\mathbf{x} = (\mathbf{A} - \lambda\mathbf{B})\mathbf{x} = 0 \quad (2.22)$$

The values of λ are determined so that

$$\det(\mathbf{C}) = \det(\mathbf{A} - \lambda\mathbf{B}) = 0 \quad (2.23)$$

The corresponding values of λ are the *eigenvalues*.

The homogeneous system of equations is generally written in the form

$$\mathbf{Ax} = \lambda\mathbf{Bx} \quad (2.24)$$

In many problems $\mathbf{B} = \mathbf{I}$, and Eq. (2.24) becomes

$$\boxed{\mathbf{Ax} = \lambda\mathbf{x}} \quad (2.25)$$

In problems where $\mathbf{B} \neq \mathbf{I}$, define the matrix $\bar{\mathbf{A}} = (\mathbf{B}^{-1}\mathbf{A})$. Then Eq. (2.24) becomes

$$\boxed{\bar{\mathbf{A}}\mathbf{x} = \lambda\mathbf{x}} \quad (2.26)$$

which has the same form as Eq. (2.25). Equation (2.25) can be written in the alternate form

$$\boxed{(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0} \quad (2.27)$$

which is the most common form of an eigenproblem statement.

The eigenvalues can be found by expanding $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ and finding the roots of the resulting n th-order polynomial, which is called the *characteristic equation*. This procedure is illustrated in the following discussion.

Consider the dynamic spring-mass problem specified by Eq. (2.9):

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \begin{bmatrix} (8 - \lambda) & -2 & -2 \\ -2 & (4 - \lambda) & -2 \\ -2 & -2 & (13 - \lambda) \end{bmatrix} \mathbf{x} = 0 \quad (2.28)$$

The characteristic equation, $|\mathbf{A} - \lambda\mathbf{I}| = 0$, is

$$(8 - \lambda)[(4 - \lambda)(13 - \lambda) - 4] - (-2)[(-2)(13 - \lambda) - 4] + (-2)[4 + 2(4 - \lambda)] = 0 \quad (2.29)$$

$$\lambda^3 - 25\lambda^2 + 176\lambda - 300 = 0 \quad (2.30)$$

The eigenvalues are

$$\lambda = 13.870585, \quad 8.620434, \quad 2.508981 \quad (2.31)$$

which can be demonstrated by direct substitution. From Eq. (2.7), the corresponding natural frequencies of oscillation, in terms of $f = 2\pi\omega$, where $\omega^2 = \lambda K_{\text{ref}}/m_{\text{ref}}$, are

$$f_1 = 2\pi\omega_1 = 2\pi\sqrt{\frac{\lambda_1 K_{\text{ref}}}{m_{\text{ref}}}} = 2\pi\sqrt{\frac{(13.870585)(10)}{2}} = 16.656 \text{ Hz} \quad (2.32a)$$

$$f_2 = 2\pi\omega_2 = 2\pi\sqrt{\frac{\lambda_2 K_{\text{ref}}}{m_{\text{ref}}}} = 2\pi\sqrt{\frac{(8.620434)(10)}{2}} = 13.130 \text{ Hz} \quad (2.32b)$$

$$f_3 = 2\pi\omega_3 = 2\pi\sqrt{\frac{\lambda_3 K_{\text{ref}}}{m_{\text{ref}}}} = 2\pi\sqrt{\frac{(2.508981)(10)}{2}} = 7.084 \text{ Hz} \quad (2.32c)$$

where Hz = Hertz = 1.0 cycle/sec.

The eigenvectors corresponding to λ_1 to λ_3 are determined as follows. For each eigenvalue λ_i ($i = 1, 2, 3$), find the amplitudes X_2 and X_3 relative to the amplitude X_1 by letting $X_1 = 1.0$. Any two of the three equations given by Eq. (2.9) can be used to solve for X_2 and X_3 with $X_1 = 1.0$. From Eqs. (2.9a) and (2.9c),

$$(8 - \lambda)X_1 - 2X_2 - 2X_3 = 0 \quad (2.33a)$$

$$-2X_1 - 2X_2 + (13 - \lambda)X_3 = 0 \quad (2.33b)$$

Solving Eqs. (2.33a) and (2.33b) for X_3 and substituting that result in Eq. (2.33a) yields

$$X_3 = \frac{(10 - \lambda)}{15} - \lambda \quad \text{and} \quad X_2 = \frac{(8 - \lambda)}{2} - X_3 \quad (2.33c)$$

Substituting λ_1 to λ_3 into Eq. (2.33c) yields:

For $\lambda_1 = 13.870586$:

$$\mathbf{X}_1 = [1.000000 \quad 0.491779 \quad -3.427072] \quad (2.34a)$$

For $\lambda_2 = 8.620434$:

$$\mathbf{X}_2 = [1.000000 \quad -0.526465 \quad 0.216247] \quad (2.34b)$$

For $\lambda_3 = 2.508981$:

$$\mathbf{X}_3 = [1.000000 \quad 2.145797 \quad 0.599712] \quad (2.34c)$$

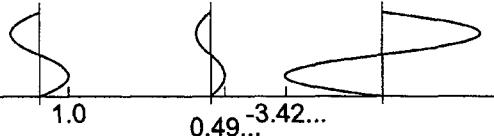
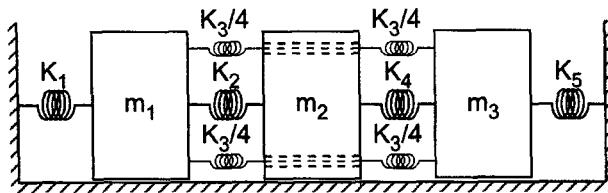
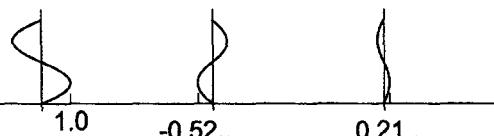
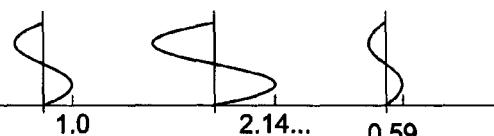
The modes of oscillation corresponding to these results are illustrated in Figure 2.4.

In summary, eigenproblems arise from homogeneous systems of equations that contain an unspecified arbitrary parameter in the coefficients. The *characteristic equation* is determined by expanding the determinant

$$\det(\mathbf{A} - \lambda \mathbf{I}) = 0 \quad (2.35)$$

which yields an n th-degree polynomial in λ . Solving the characteristic equation yields n eigenvalues λ_i ($i = 1, 2, \dots, n$). The n eigenvectors \mathbf{x}_i ($i = 1, 2, \dots, n$), corresponding to the n eigenvalues λ_i ($i = 1, 2, \dots, n$) are found by substituting the individual eigenvalues into the homogeneous system of equations, which is then solved for the eigenvectors.

In principle, the solution of eigenproblems is straightforward. In practice, when the size of the system of equations is very large, expanding the characteristic determinant to obtain the characteristic equation is difficult. Solving high-degree polynomials for the eigenvalues presents yet another difficult problem. Consequently, more straightforward

(a) $f_1 = 16.656 \text{ Hz.}$ (b) $f_2 = 13.130 \text{ Hz.}$ (c) $f_3 = 7.084 \text{ Hz.}$ **Figure 2.4** Mode shapes.

procedures for solving eigenproblems are desired. An iterative numerical procedure, called the *power method*, and its variations are presented in Section 2.3 to illustrate the numerical solution of eigenproblems. The direct method is presented in Section 2.4. The most general method, the QR method, is presented in Section 2.5.

2.3 THE POWER METHOD

Consider the linear eigenproblem:

$$\boxed{\mathbf{Ax} = \lambda \mathbf{x}} \quad (2.36)$$

The *power method* is based on repetitive multiplication of a trial eigenvector $\mathbf{x}^{(0)}$ by matrix \mathbf{A} with a scaling of the resulting vector \mathbf{y} , so that the scaling factor approaches the largest eigenvalue λ and the scaled \mathbf{y} vector approaches the corresponding eigenvector \mathbf{x} . The power method and several of its variations are presented in this section.

2.3.1. The Direct Power Method

When the largest (in absolute value) eigenvalue of \mathbf{A} is distinct, its value can be found using an iterative technique called the *direct power method*. The procedure is as follows:

1. Assume a trial value $\mathbf{x}^{(0)}$ for the eigenvector \mathbf{x} . Choose one component of \mathbf{x} to be unity. Designate that component as the *unity* component.
2. Perform the matrix multiplication:

$$\mathbf{Ax}^{(0)} = \mathbf{y}^{(1)} \quad (2.37)$$

3. Scale $\mathbf{y}^{(1)}$ so that the unity component remains unity:

$$\mathbf{y}^{(1)} = \lambda^{(1)} \mathbf{x}^{(1)} \quad (2.38)$$

4. Repeat steps 2 and 3 with $\mathbf{x} = \mathbf{x}^{(1)}$. Iterate to convergence. At convergence, the value λ is the largest (in absolute value) eigenvalue of \mathbf{A} , and the vector \mathbf{x} is the corresponding eigenvector (scaled to unity on the unity component).

The general algorithm for the power method is as follows:

$$\boxed{\mathbf{Ax}^{(k)} = \mathbf{y}^{(k+1)} = \lambda^{(k+1)} \mathbf{x}^{(k+1)}}$$

(2.39)

When the iterations indicate that the unity component could be zero, a different unity component must be chosen. The method is slow to converge when the magnitudes (in absolute value) of the largest eigenvalues are nearly the same. When the largest eigenvalues are of equal magnitude, the power method, as described, fails.

Example 2.1. The direct power method.

Find the largest (in absolute value) eigenvalue and the corresponding eigenvector of the matrix given by Eq. (2.11):

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \quad (2.40)$$

Assume $\mathbf{x}^{(0)T} = [1.0 \quad 1.0 \quad 1.0]$. Scale the third component x_3 to unity. Then apply Eq. (2.39).

$$\mathbf{Ax}^{(0)} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} 4.00 \\ 0.00 \\ 9.00 \end{bmatrix} \quad \lambda^{(1)} = 9.00 \quad \mathbf{x}^{(1)} = \begin{bmatrix} 0.444444 \\ 0.000000 \\ 1.000000 \end{bmatrix} \quad (2.41)$$

$$\mathbf{Ax}^{(1)} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \begin{bmatrix} 0.444444 \\ 0.000000 \\ 1.000000 \end{bmatrix} = \begin{bmatrix} 1.555555 \\ -2.888888 \\ 12.111111 \end{bmatrix}$$

$$\lambda^{(2)} = 12.111111 \quad \mathbf{x}^{(2)} = \begin{bmatrix} 0.128440 \\ -0.238532 \\ 1.000000 \end{bmatrix} \quad (2.42)$$

Table 2.1. The Power Method

k	λ	x_1	x_2	x_3
0		1.000000	1.000000	1.000000
1	9.000000	0.444444	0.000000	1.000000
2	12.111111	0.128440	-0.238532	1.000000
3	13.220183	-0.037474	-0.242887	1.000000
4	13.560722	-0.133770	-0.213602	1.000000
5	13.694744	-0.192991	-0.188895	1.000000
.....				
29	13.870583	-0.291793	-0.143499	1.000000
30	13.870584	-0.291794	-0.143499	1.000000

The results of the first two iterations presented above and subsequent iterations are presented in Table 2.1. These results were obtained on a 13-digit precision computer. The iterations were continued until λ changed by less than 0.000001 between iterations. The final solution for the largest eigenvalue, denoted as λ_1 , and the corresponding eigenvector \mathbf{x}_1 is

$$\lambda_1 = 13.870584 \quad \text{and} \quad \mathbf{x}_1^T = [-0.291794 \quad -0.143499 \quad 1.000000] \quad (2.43)$$

This problem converged very slowly (30 iterations), which is a large number of iterations for a 3×3 matrix. A procedure for accelerating the convergence of a slowly converging eigenproblem is presented in Example 2.5.

2.3.2. Basis of the Power Method

The basis of the power method is as follows. Assume that \mathbf{A} is an $n \times n$ nonsingular matrix having n eigenvalues, $\lambda_1, \lambda_2, \dots, \lambda_n$, with n corresponding linearly independent eigenvectors, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$. Assume further that $|\lambda_1| > |\lambda_2| > \dots > |\lambda_n|$. Since the eigenvectors, \mathbf{x}_i ($i = 1, 2, \dots, n$), are linearly independent (i.e., they span the n -dimensional space), any arbitrary vector \mathbf{x} can be expressed as a linear combination of the eigenvectors. Thus,

$$\mathbf{x} = C_1 \mathbf{x}_1 + C_2 \mathbf{x}_2 + \dots + C_n \mathbf{x}_n = \sum_{i=1}^n C_i \mathbf{x}_i \quad (2.44)$$

Multiplying both sides of Eq. (2.44) by \mathbf{A} , $\mathbf{A}^2, \dots, \mathbf{A}^k$, etc., where the superscript denotes repetitive matrix multiplication, and recalling that $\mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i$, yields

$$\mathbf{Ax} = \sum_{i=1}^n C_i \mathbf{Ax}_i = \sum_{i=1}^n C_i \lambda_i \mathbf{x}_i = \mathbf{y}^{(1)} \quad (2.45)$$

$$\mathbf{A}^2 \mathbf{x} = \mathbf{Ay}^{(1)} = \sum_{i=1}^n C_i \lambda_i \mathbf{Ax}_i = \sum_{i=1}^n C_i \lambda_i^2 \mathbf{x}_i = \mathbf{y}^{(2)} \quad (2.46)$$

$$\mathbf{A}^k \mathbf{x} = \mathbf{Ay}^{(k-1)} = \sum_{i=1}^n C_i \lambda_i^{k-1} \mathbf{Ax}_i = \sum_{i=1}^n C_i \lambda_i^k \mathbf{x}_i = \mathbf{y}^{(k)} \quad (2.47)$$

Factoring λ_1^k out of the next to last term in Eq. (2.47) yields

$$\mathbf{A}^k \mathbf{x} = \lambda_1^k \sum_{i=1}^n C_i \left(\frac{\lambda_i}{\lambda_1} \right)^k \mathbf{x}_i = \mathbf{y}^{(k)} \quad (2.48)$$

Since $|\lambda_1| > |\lambda_i|$ for $i = 2, 3, \dots, n$, the ratios $(\lambda_i/\lambda_1)^k \rightarrow 0$ as $k \rightarrow \infty$, and Eq. (2.48) approaches the limit

$$\mathbf{A}^k \mathbf{x} = \lambda_1^k C_1 \mathbf{x}_1 = \mathbf{y}^{(k)} \quad (2.49)$$

Equation (2.49) approaches zero if $|\lambda_1| < 1$ and approaches infinity if $|\lambda_1| > 1$. Thus, Eq. (2.49) must be scaled between steps.

Scaling can be accomplished by scaling any component of vector $\mathbf{y}^{(k)}$ to unity at each step in the process. Choose the first component of vector $\mathbf{y}^{(k)}$, $y_1^{(k)}$, to be that component. Thus, $x_1 = 1.0$, and the first component of Eq. (2.49) is

$$y_1^{(k)} = \lambda_1^k C_1 \quad (2.50)$$

Applying Eq. (2.49) one more time (i.e., from k to $k+1$) yields

$$y_1^{(k+1)} = \lambda_1^{k+1} C_1 \quad (2.51)$$

Taking the ratio of Eq. (2.51) to Eq. (2.50) gives

$$\frac{y_1^{(k+1)}}{y_1^{(k)}} = \frac{\lambda_1^{k+1} C_1}{\lambda_1^k C_1} = \lambda_1 \quad (2.52)$$

Thus, if $y_1^{(k)} = 1$, then $y_1^{(k+1)} = \lambda_1$. If $y_1^{(k+1)}$ is scaled by λ_1 so that $y_1^{(k+1)} = 1$, then $y_1^{(k+2)} = \lambda_1$, etc. Consequently, scaling a particular component of vector \mathbf{y} each iteration essentially factors λ_1 out of vector \mathbf{y} , so that Eq. (2.49) converges to a finite value. In the limit as $k \rightarrow \infty$, the scaling factor approaches λ_1 , and the scaled vector \mathbf{y} approaches the eigenvector \mathbf{x}_1 .

Several restrictions apply to the power method.

1. The largest eigenvalue must be distinct.
2. The n eigenvectors must be independent.
3. The initial guess $\mathbf{x}_i^{(0)}$ must contain some component of eigenvector \mathbf{x}_i , so that $C_i \neq 0$.
4. The convergence rate is proportional to the ratio

$$\frac{|\lambda_i|}{|\lambda_{i-1}|}$$

where λ_i is the largest (in magnitude) eigenvalue and λ_{i-1} is the second largest (in magnitude) eigenvalue.

2.3.3. The Inverse Power Method

When the smallest (in absolute value) eigenvalue of matrix \mathbf{A} is distinct, its value can be found using a variation of the power method called the *inverse power method*. Essentially, this involves finding the largest (in magnitude) eigenvalue of the inverse matrix \mathbf{A}^{-1} , which is the smallest (in magnitude) eigenvalue of matrix \mathbf{A} . Recall the original eigenproblem:

$$\mathbf{Ax} = \lambda \mathbf{x} \quad (2.53)$$

Multiplying Eq. (2.53) by \mathbf{A}^{-1} gives

$$\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{Ix} = \mathbf{x} = \lambda\mathbf{A}^{-1}\mathbf{x} \quad (2.54)$$

Rearranging Eq. (2.54) yields an eigenproblem for \mathbf{A}^{-1} . Thus,

$$\boxed{\mathbf{A}^{-1}\mathbf{x} = \left(\frac{1}{\lambda}\right)\mathbf{x} = \lambda_{\text{inverse}}\mathbf{x}} \quad (2.55)$$

The eigenvalues of matrix \mathbf{A}^{-1} , that is, λ_{inverse} , are the reciprocals of the eigenvalues of matrix \mathbf{A} . The eigenvectors of matrix \mathbf{A}^{-1} are the same as the eigenvectors of matrix \mathbf{A} . The power method can be used to find the largest (in absolute value) eigenvalue of matrix \mathbf{A}^{-1} , λ_{inverse} . The reciprocal of that eigenvalue is the smallest (in absolute value) eigenvalue of matrix \mathbf{A} .

In practice the LU method is used to solve the inverse eigenproblem instead of calculating the inverse matrix \mathbf{A}^{-1} . The power method applied to matrix \mathbf{A}^{-1} is given by

$$\mathbf{A}^{-1}\mathbf{x}^{(k)} = \mathbf{y}^{(k+1)} \quad (2.56)$$

Multiplying Eq. (2.56) by \mathbf{A} gives

$$\mathbf{AA}^{-1}\mathbf{x}^{(k)} = \mathbf{Ix}^{(k)} = \mathbf{x}^{(k)} = \mathbf{Ay}^{(k+1)} \quad (2.57)$$

which can be written as

$$\boxed{\mathbf{Ay}^{(k+1)} = \mathbf{x}^{(k)}} \quad (2.58)$$

Equation (2.58) is in the standard form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x} = \mathbf{y}^{(k+1)}$ and $\mathbf{b} = \mathbf{x}^{(k)}$. Thus, for a given $\mathbf{x}^{(k)}$, $\mathbf{y}^{(k+1)}$ can be found by the Doolittle LU method. The procedure is as follows:

1. Solve for \mathbf{L} and \mathbf{U} such that $\mathbf{LU} = \mathbf{A}$ by the Doolittle LU method.
2. Assume $\mathbf{x}^{(0)}$. Designate a component of \mathbf{x} to be unity.
3. Solve for \mathbf{x}' by forward substitution using the equation

$$\mathbf{Lx}' = \mathbf{x}^{(0)} \quad (2.59)$$

4. Solve for $\mathbf{y}^{(1)}$ by back substitution using the equation

$$\mathbf{Uy}^{(1)} = \mathbf{x}' \quad (2.60)$$

5. Scale $\mathbf{y}^{(1)}$ so that the unity component is unity. Thus,

$$\mathbf{y}^{(1)} = \lambda_{\text{inverse}}^{(1)}\mathbf{x}^{(1)} \quad (2.61)$$

6. Repeat steps 3 to 5 with $\mathbf{x}^{(1)}$. Iterate to convergence. At convergence, $\lambda = 1/\lambda_{\text{inverse}}$, and $\mathbf{x}^{(k+1)}$ is the corresponding eigenvector.

The inverse power method algorithm is as follows:

$$\boxed{\mathbf{Lx}' = \mathbf{x}^{(k)}} \quad (2.62)$$

$$\boxed{\mathbf{Uy}^{(k+1)} = \mathbf{x}'} \quad (2.63)$$

$$\boxed{\mathbf{y}^{(k+1)} = \lambda_{\text{inverse}}^{(k+1)}\mathbf{x}^{(k+1)}} \quad (2.64)$$

Example 2.2. The inverse power method.

Find the smallest (in absolute value) eigenvalue and the corresponding eigenvector of the matrix given by Eq. (2.11):

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \quad (2.65)$$

Assume $\mathbf{x}^{(0)T} = [1.0 \quad 1.0 \quad 1.0]$. Scale the first component of \mathbf{x} to unity. The first step is to solve for \mathbf{L} and \mathbf{U} by the Doolittle LU method. The results are

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} 8 & -2 & -2 \\ 0 & 7/2 & -5/2 \\ 0 & 0 & 75/7 \end{bmatrix} \quad (2.66)$$

Solve for \mathbf{x}' by forward substitution using $\mathbf{L}\mathbf{x}' = \mathbf{x}^{(0)}$.

$$\begin{bmatrix} 1 & 0 & 0 \\ -1/4 & 1 & 0 \\ -1/4 & -5/7 & 1 \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}$$

$$x'_1 = 1.0$$

$$x'_2 = 1.0 - (-1/4)(1.0) = 5/4$$

$$x'_3 = 1.0 - (-1/4)(1.0) - (-5/7)(5/4) = 15/7 \quad (2.67)$$

Solve for $\mathbf{y}^{(1)}$ by back substitution using $\mathbf{U}\mathbf{y}^{(1)} = \mathbf{x}'$.

$$\begin{bmatrix} 8 & -2 & -2 \\ 0 & 7/2 & -5/2 \\ 0 & 0 & 75/7 \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 5/4 \\ 15/7 \end{bmatrix}$$

$$y_1^{(1)} = [1.0 - (-2.0)(0.5) - (-2)(0.2)]/8 = 0.30$$

$$y_2^{(1)} = [5/4 - (-5/2)(0.2)]/(7/2) = 0.50$$

$$y_3^{(1)} = (15/7)/(75/7) = 0.20 \quad (2.68)$$

Scale $\mathbf{y}^{(1)}$ so that the unity component is unity.

$$\mathbf{y}^{(1)} = \begin{bmatrix} 0.30 \\ 0.50 \\ 0.20 \end{bmatrix} \quad \lambda_{\text{inverse}}^{(1)} = 0.300000 \quad \mathbf{x}^{(1)} = \begin{bmatrix} 1.000000 \\ 1.666667 \\ 0.666667 \end{bmatrix} \quad (2.69)$$

The results of the first iteration presented above and subsequent iterations are presented in Table 2.2. These results were obtained on a 13-digit precision computer. The iterations were continued until λ_{inverse} changed by less than 0.000001 between iterations.

The final solution for the smallest eigenvalue λ_3 and the corresponding eigenvector \mathbf{x}_3 is

$$\lambda_3 = \frac{1}{\lambda_{\text{inverse}}} = \frac{1}{0.398568} = 2.508981 \quad \text{and}$$

$$\mathbf{x}_3^T = [1.000000 \quad 2.145797 \quad 0.599712] \quad (2.70)$$

Table 2.2. The Inverse Power Method

k	λ_{inverse}	x_1	x_2	x_3
0		1.000000	1.000000	1.000000
1	0.300000	1.000000	1.666667	0.666667
2	0.353333	1.000000	1.981132	0.603774
3	0.382264	1.000000	2.094439	0.597565
4	0.393346	1.000000	2.130396	0.598460
.....				
12	0.398568	1.000000	2.145796	0.599712
13	0.398568	1.000000	2.145797	0.599712

2.3.4. The Shifted Power Method

The eigenvalues of a matrix \mathbf{A} may be shifted by a scalar s by subtracting $s\mathbf{I}\mathbf{x} = s\mathbf{x}$ from both sides of the standard eigenproblem, $\mathbf{Ax} = \lambda\mathbf{x}$. Thus,

$$\mathbf{Ax} - s\mathbf{Ix} = \lambda\mathbf{x} - s\mathbf{x} \quad (2.71)$$

which yields

$$(\mathbf{A} - s\mathbf{I})\mathbf{x} = (\lambda - s)\mathbf{x} \quad (2.72)$$

which can be written as

$$\boxed{\mathbf{A}_{\text{shifted}}\mathbf{x} = \lambda_{\text{shifted}}\mathbf{x}} \quad (2.73)$$

where $\mathbf{A}_{\text{shifted}} = (\mathbf{A} - s\mathbf{I})$ is the shifted matrix and $\lambda_{\text{shifted}} = \lambda - s$ is the eigenvalue of the shifted matrix. Shifting a matrix \mathbf{A} by a scalar, s , shifts the eigenvalues by s . Shifting a matrix by a scalar does not affect the eigenvectors. Shifting the eigenvalues of a matrix can be used to:

1. Find the *opposite extreme eigenvalue*, which is either the smallest (in absolute value) eigenvalue or the largest (in absolute value) eigenvalue of opposite sign
2. Find *intermediate eigenvalues*
3. *Accelerate convergence* for slowly converging eigenproblems

2.3.4.1. Shifting Eigenvalues to Find the Opposite Extreme Eigenvalue

Consider a matrix whose eigenvalues are all the same sign, for example 1, 2, 4, and 8. For this matrix, 8 is the largest (in absolute value) eigenvalue and 1 is the opposite extreme eigenvalue. Solve for the largest (in absolute value) eigenvalue, $\lambda_{\text{Largest}} = 8$, by the direct power method. Shifting the eigenvalues by $s = 8$ yields the shifted eigenvalues $-7, -6, -4$, and 0. Solve for the largest (in absolute value) eigenvalue of the shifted matrix, $\lambda_{\text{shifted,Largest}} = -7$, by the power method. Then $\lambda_{\text{Smallest}} = \lambda_{\text{shifted,Largest}} + 8 = -7 + 8 = 1$. This procedure yields the same eigenvalue as the inverse power method applied to the original matrix.

Consider a matrix whose eigenvalues are both positive and negative, for example, $-1, 2, 4$, and 8. For this matrix, 8 is the largest (in absolute value) eigenvalue and -1 is the opposite extreme eigenvalue. Solve for the largest (in absolute value) eigenvalue,

$\lambda_{\text{Largest}} = 8$, by the power method. Shifting the eigenvalues by $s = 8$ yields the shifted eigenvalues $-9, -6, -4$, and 0 . Solve for the largest (in absolute value) eigenvalue of the shifted matrix, $\lambda_{\text{shifted,Largest}} = -9$, by the power method. Then $\lambda_{\text{Largest,Negative}} = \lambda_{\text{shifted,Largest}} + 8 = -9 + 8 = -1$.

Both of the cases described above are solved by shifting the matrix by the largest (in absolute value) eigenvalue and applying the direct power method to the shifted matrix. Generally speaking, it is not known a priori which result will be obtained. If all the eigenvalues of a matrix have the same sign, the smallest (in absolute value) eigenvalue will be obtained. If a matrix has both positive and negative eigenvalues, the largest eigenvalue of opposite sign will be obtained.

The above procedure is called the *shifted direct power method*. The procedure is as follows:

1. Solve for the largest (in absolute value) eigenvalue λ_{Largest} .
2. Shift the eigenvalues of matrix \mathbf{A} by $s = \lambda_{\text{Largest}}$ to obtain the shifted matrix $\mathbf{A}_{\text{shifted}}$.
3. Solve for the eigenvalue λ_{shifted} of the shifted matrix $\mathbf{A}_{\text{shifted}}$ by the direct power method.
4. Calculate the opposite extreme eigenvalue of matrix \mathbf{A} by $\lambda = \lambda_{\text{shifted}} + s$.

Example 2.3. The shifted direct power method for opposite extreme eigenvalues.

Find the opposite extreme eigenvalue of matrix \mathbf{A} by shifting the eigenvalues by $s = \lambda_{\text{Largest}} = 13.870584$. The original and shifted matrices are:

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \quad (2.74)$$

$$\begin{aligned} \mathbf{A}_{\text{shifted}} &= \begin{bmatrix} (8 - 13.870584) & -2 & -2 \\ -2 & (4 - 13.870584) & -2 \\ -2 & -2 & (13 - 13.870584) \end{bmatrix} \\ &= \begin{bmatrix} -5.870584 & -2.000000 & -2.000000 \\ -2.000000 & -9.870584 & -2.000000 \\ -2.000000 & -2.000000 & -0.870584 \end{bmatrix} \end{aligned} \quad (2.75)$$

Assume $\mathbf{x}^{(0)T} = [1.0 \ 1.0 \ 1.0]$. Scale the second component to unity. Applying the power method to matrix $\mathbf{A}_{\text{shifted}}$ gives

$$\begin{aligned} \mathbf{A}_{\text{shifted}} \mathbf{x}^{(0)} &= \begin{bmatrix} -5.870584 & -2.000000 & -2.000000 \\ -2.000000 & -9.870584 & -2.000000 \\ -2.000000 & -2.000000 & -0.870584 \end{bmatrix} \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} \\ &= \begin{bmatrix} -9.870584 \\ -13.870584 \\ -4.870584 \end{bmatrix} = \mathbf{y}^{(1)} \end{aligned} \quad (2.76)$$

Table 2.3. Shifting Eigenvalues to Find the Opposite Extreme Eigenvalue

k	λ_{shifted}	x_1	x_2	x_3
0		1.000000	1.000000	1.000000
1	-13.870584	0.711620	1.000000	0.351145
2	-11.996114	0.573512	1.000000	0.310846
3	-11.639299	0.514510	1.000000	0.293629
4	-11.486864	0.488187	1.000000	0.285948
...
19	-11.361604	0.466027	1.000000	0.279482
20	-11.361604	0.466027	1.000000	0.279482

Scaling the unity component of $\mathbf{y}^{(1)}$ to unity gives

$$\lambda_{\text{shifted}}^{(1)} = -13.870584 \quad \text{and} \quad \mathbf{x}^{(1)} = \begin{bmatrix} 0.711620 \\ 1.000000 \\ 0.351145 \end{bmatrix} \quad (2.77)$$

The results of the first iteration presented above and subsequent iterations are presented in Table 2.3. These results were obtained on a 13-digit precision computer with an absolute convergence tolerance of 0.000001.

The largest (in magnitude) eigenvalue of $\mathbf{A}_{\text{shifted}}$ is $\lambda_{\text{shifted,Largest}} = -11.361604$. Thus, the opposite extreme eigenvalue of matrix \mathbf{A} is

$$\lambda = \lambda_{\text{shifted,Largest}} + 13.870584 = -11.361604 + 13.870586 = 2.508980 \quad (2.78)$$

Since this eigenvalue, $\lambda = 2.508980$, has the same sign as the largest (in absolute value) eigenvalue, $\lambda = 13.870584$, it is the smallest (in absolute value) eigenvalue of matrix \mathbf{A} , and all the eigenvalues of matrix \mathbf{A} are positive.

2.3.4.2. Shifting Eigenvalues to Find Intermediate Eigenvalues

Intermediate eigenvalues λ_{Inter} lie between the largest eigenvalue and the smallest eigenvalue. Consider a matrix whose eigenvalues are 1, 2, 4, and 8. Solve for the largest (in absolute value) eigenvalue, $\lambda_{\text{Largest}} = 8$, by the power method and the smallest eigenvalue, $\lambda_{\text{Smallest}} = 1$, by the inverse power method. Two intermediate eigenvalues, $\lambda_{\text{Inter}} = 2$ and 4, remain to be determined. If λ_{Inter} is guessed to be $\lambda_{\text{Guess}} = 5$ and the eigenvalues are shifted by $s = 5$, the eigenvalues of the shifted matrix are -4, -3, -1, and 3. Applying the inverse power method to the shifted matrix gives $\lambda_{\text{shifted}} = -1$, from which $\lambda = \lambda_{\text{shifted}} + s = -1 + 5 = 4$. The power method is not an efficient method for finding intermediate eigenvalues. However, it can be used for that purpose.

The above procedure is called the *shifted inverse power method*. The procedure as follows:

1. Guess a value λ_{Guess} for the intermediate eigenvalue of the shifted matrix.
2. Shift the eigenvalues by $s = \lambda_{\text{Guess}}$ to obtain the shifted matrix $\mathbf{A}_{\text{shifted}}$.
3. Solve for the eigenvalue $\lambda_{\text{shifted,inverse}}$ of the inverse shifted matrix $\mathbf{A}_{\text{shifted}}^{-1}$ by the inverse power method applied to matrix $\mathbf{A}_{\text{shifted}}$.

4. Solve for $\lambda_{\text{shifted}} = 1/\lambda_{\text{shifted,inverse}}$.
5. Solve for the intermediate eigenvalue $\lambda_{\text{inter}} = \lambda_{\text{shifted}} + s$.

Example 2.4. The shifted inverse power method for intermediate eigenvalues.

Let's attempt to find an intermediate eigenvalue of matrix \mathbf{A} by guessing its value, for example, $\lambda_{\text{Guess}} = 10.0$. The corresponding shifted matrix $\mathbf{A}_{\text{shifted}}$ is

$$\begin{aligned}\mathbf{A}_{\text{shifted}} &= (\mathbf{A} - \lambda_{\text{Guess}} \mathbf{I}) = \begin{bmatrix} (8 - 10.0) & -2 & -2 \\ -2 & (4 - 10.0) & -2 \\ -2 & -2 & (13 - 10.0) \end{bmatrix} \\ &= \begin{bmatrix} -2.0 & -2.0 & -2.0 \\ -2.0 & -6.0 & -2.0 \\ -2.0 & -2.0 & 3.0 \end{bmatrix}\end{aligned}\quad (2.79)$$

Solving for \mathbf{L} and \mathbf{U} by the Doolittle LU method yields:

$$\mathbf{L} = \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 1.0 \end{bmatrix} \quad \text{and} \quad \mathbf{U} = \begin{bmatrix} -2.0 & -2.0 & -2.0 \\ 0.0 & -4.0 & 0.0 \\ 0.0 & 0.0 & 5.0 \end{bmatrix}\quad (2.80)$$

Assume $\mathbf{x}^{(0)T} = [1.0 \ 1.0 \ 1.0]$. Scale the first component of \mathbf{x} to unity. Solve for \mathbf{x}' by forward substitution using $\mathbf{L}\mathbf{x}' = \mathbf{x}^{(0)}$:

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 1.0 \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}\quad (2.81)$$

which yields

$$x'_1 = 1.0 \quad (2.82a)$$

$$x'_2 = 1.0 - 1.0(1.0) = 0.0 \quad (2.82b)$$

$$x'_3 = 1.0 - 1.0(1.0) - 1.0(0.0) = 0.0 \quad (2.82c)$$

Solve for $\mathbf{y}^{(1)}$ by back substitution using $\mathbf{U}\mathbf{y}^{(1)} = \mathbf{x}'$.

$$\begin{bmatrix} -2.0 & -2.0 & -2.0 \\ 0.0 & -4.0 & 0.0 \\ 0.0 & 0.0 & 5.0 \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}\quad (2.83)$$

which yields

$$y_3^{(1)} = 0.0/(5.0) = 0.0 \quad (2.84a)$$

$$y_2^{(1)} = [0.0 - (0.0)(0.0)]/(-4.0) = 0.0 \quad (2.84b)$$

$$y_1^{(1)} = [1.0 - (-2.0)(0.0) - (-2.0)(0.0)]/(-2.0) = -0.50 \quad (2.84c)$$

Table 2.4. Shifting Eigenvalues to Find Intermediate Eigenvalues

k	$\lambda_{\text{shifted,inverse}}$	x_1	x_2	x_3
0		1.000000	1.000000	1.000000
1	-0.500000	1.000000	0.000000	0.000000
2	-0.550000	1.000000	-0.454545	0.363636
3	-0.736364	1.000000	-0.493827	0.172840
4	-0.708025	1.000000	-0.527463	0.233653
.....				
14	-0.724865	1.000000	-0.526465	0.216248
15	-0.724866	1.000000	-0.526465	0.216247

Scale $\mathbf{y}^{(1)}$ so that the unity component is unity.

$$\mathbf{y}^{(1)} = \begin{bmatrix} -0.50 \\ 0.00 \\ 0.00 \end{bmatrix} \quad \lambda_{\text{shifted,inverse}}^{(1)} = -0.50 \quad \mathbf{x}^{(1)} = \begin{bmatrix} 1.00 \\ 0.00 \\ 0.00 \end{bmatrix} \quad (2.85)$$

The first iteration and subsequent iterations are summarized in Table 2.4. These results were obtained on a 13-digit precision computer with an absolute convergence tolerance of 0.000001.

Thus, the largest (in absolute value) eigenvalue of matrix $\mathbf{A}_{\text{shifted}}^{-1}$ is $\lambda_{\text{shifted,inverse}} = -0.724866$. Consequently, the corresponding eigenvalue of matrix $\mathbf{A}_{\text{shifted}}$ is

$$\lambda_{\text{shifted}} = \frac{1}{\lambda_{\text{shifted,inverse}}} = \frac{1}{-0.724866} = -1.379566 \quad (2.86)$$

Thus, the intermediate eigenvalue of matrix \mathbf{A} is

$$\lambda_I = \lambda_{\text{shifted}} + s = -1.379566 + 10.000000 = 8.620434 \quad (2.87)$$

and the corresponding eigenvector is $\mathbf{x}^T = [1.0 \quad -0.526465 \quad 0.216247]$.

2.3.4.3. Shifting Eigenvalues to Accelerate Convergence

The shifting eigenvalue concept can be used to accelerate the convergence of the power method for a slowly converging eigenproblem. When an estimate λ_{Est} of an eigenvalue of matrix \mathbf{A} is known, for example, from several initial iterations of the direct power method, the eigenvalues can be shifted by this approximate value so that the shifted matrix has an eigenvalue near zero. This eigenvalue can then be found by the inverse power method.

The above procedure is called the *shifted inverse power method*. The procedure is as follows:

1. Obtain an estimate λ_{Est} of the eigenvalue λ , for example, by several applications of the direct power method.
2. Shift the eigenvalues by $s = \lambda_{\text{Est}}$ to obtain the shifted matrix, $\mathbf{A}_{\text{shifted}}$.
3. Solve for the eigenvalue $\lambda_{\text{shifted,inverse}}$ of the inverse shifted matrix $\mathbf{A}_{\text{shifted}}^{-1}$ by the inverse power method applied to matrix $\mathbf{A}_{\text{shifted}}$. Let the first guess for \mathbf{x} be the value of \mathbf{x} corresponding to λ_{Est} .

4. Solve for $\lambda_{\text{shifted}} = 1/\lambda_{\text{shifted,inverse}}$.
5. Solve for $\lambda = \lambda_{\text{shifted}} + s$.

Example 2.5. The shifted inverse power method for accelerating convergence.

The first example of the power method, Example 2.1, converged very slowly since the two largest eigenvalues of matrix \mathbf{A} are close together (i.e., 13.870584 and 8.620434). Convergence can be accelerated by using the results of an early iteration, say iteration 5, to shift the eigenvalues by the approximate eigenvalue, and then using the inverse power method on the shifted matrix to accelerate convergence. From Example 2.1, after 5 iterations, $\lambda^{(5)} = 13.694744$ and $\mathbf{x}^{(5)T} = [-0.192991 \quad -0.188895 \quad 1.000000]$. Thus, shift matrix \mathbf{A} by $s = 13.694744$:

$$\mathbf{A}_{\text{shifted}} = (\mathbf{A} - s\mathbf{I}) = \begin{bmatrix} -5.694744 & -2.000000 & -2.00000 \\ -2.000000 & -9.694744 & -2.00000 \\ -2.000000 & -2.000000 & -0.694744 \end{bmatrix} \quad (2.88)$$

The corresponding \mathbf{L} and \mathbf{U} matrices are

$$\mathbf{L} = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 \\ 0.351201 & 1.000000 & 0.000000 \\ 0.351201 & 0.144300 & 1.000000 \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} -5.694744 & -2.000000 & -2.000000 \\ 0.000000 & -8.992342 & -1.297598 \\ 0.000000 & 0.000000 & 0.194902 \end{bmatrix} \quad (2.89)$$

Let $\mathbf{x}^{(0)T} = \mathbf{x}^{(5)T}$ and continue scaling the third component of \mathbf{x} to unity. Applying the inverse power method to matrix $\mathbf{A}_{\text{shifted}}$ yields the results presented in Table 2.5. These results were obtained on a 13-digit precision computer with an absolute convergence tolerance of 0.000001. The eigenvalue λ_{shifted} of the shifted matrix $\mathbf{A}_{\text{shifted}}$ is

$$\lambda_{\text{shifted}} = \frac{1}{\lambda_{\text{shifted,inverse}}} = \frac{1}{5.686952} = 0.175841 \quad (2.90)$$

Table 2.5. Shifting Eigenvalues to Accelerate Convergence

k	$\lambda_{\text{shifted,inverse}}$	x_1	x_2	x_3
0		-0.192991	-0.188895	1.000000
1	5.568216	-0.295286	-0.141881	1.000000
2	5.691139	-0.291674	-0.143554	1.000000
3	5.686807	-0.291799	-0.143496	1.000000
4	5.686957	-0.291794	-0.143498	1.000000
5	5.686952	-0.291794	-0.143498	1.000000
6	5.686952	-0.291794	-0.143498	1.000000

Thus, the eigenvalue λ of the original matrix \mathbf{A} is

$$\lambda = \lambda_{\text{shifted}} + s = 0.175841 + 13.694744 = 13.870585 \quad (2.91)$$

This is the same result obtained in the first example with 30 iterations. The present solution required only 11 total iterations: 5 for the initial solution and 6 for the final solution.

2.3.5. Summary

In summary, the largest eigenvalue, $\lambda = 13.870584$, was found by the power method; the smallest eigenvalue, $\lambda = 2.508981$, was found by both the inverse power method and by shifting the eigenvalues by the largest eigenvalue; and the third (and intermediate) eigenvalue, $\lambda = 8.620434$, was found by shifting eigenvalues. The corresponding eigenvectors were also found. These results agree with the exact solution of this problem presented in Section 2.2.

2.4 THE DIRECT METHOD

The power method and its variations presented in Section 2.3 apply to linear eigenproblems of the form

$$\mathbf{Ax} = \lambda \mathbf{x} \quad (2.92)$$

Nonlinear eigenproblems of the form

$$\boxed{\mathbf{Ax} = \mathbf{B}(\lambda)\mathbf{x}} \quad (2.93)$$

where $\mathbf{B}(\lambda)$ is a nonlinear function of λ , cannot be solved by the power method. Linear eigenproblems and nonlinear eigenproblems both can be solved by a direct approach which involves finding the zeros of the characteristic equation directly.

For a linear eigenproblem, the characteristic equation is obtained from

$$\det(\mathbf{A} - \lambda \mathbf{I}) = 0 \quad (2.94)$$

Expanding Eq. (2.94), which can be time consuming for a large system, yields an n th-degree polynomial in λ . The roots of the characteristic polynomial can be determined by the methods presented in Section 3.5 for finding the roots of polynomials.

For a nonlinear eigenproblem, the characteristic equation is obtained from

$$\det[\mathbf{A} - \mathbf{B}(\lambda)] = 0 \quad (2.95)$$

Expanding Eq. (2.95) yields a nonlinear function of λ , which can be solved by the methods presented in Chapter 3.

An alternate approach for solving for the roots of the characteristic equation directly is to solve Eqs. (2.94) and (2.95) iteratively. This can be accomplished by applying the secant method, presented in Section 3.4.3, to Eqs. (2.94) and (2.95). Two initial approximations of λ are assumed, λ_0 and λ_1 , the corresponding values of the characteristic determinant are computed, and these results are used to construct a linear relationship between λ and the value of the characteristic determinant. The solution of that linear relationship is taken as the next approximation to λ , and the procedure is repeated

iteratively to convergence. Reasonable initial approximations are required, especially for nonlinear eigenproblems.

The direct method determines only the eigenvalues. The corresponding eigenvectors must be determined by substituting the eigenvalues into the system of equations and solving for the corresponding eigenvectors directly, or by applying the inverse power method one time as illustrated in Section 2.6.

Example 2.6. The direct method for a linear eigenproblem.

Let's find the largest eigenvalue of the matrix given by Eq. (2.11) by the direct method. Thus,

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \quad (2.96)$$

The characteristic determinant corresponding to Eq. (2.96) is

$$f(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I}) = \begin{bmatrix} (8 - \lambda) & -2 & -2 \\ -2 & (4 - \lambda) & -2 \\ -2 & -2 & (13 - \lambda) \end{bmatrix} = 0 \quad (2.97)$$

Equation (2.97) can be solved by the secant method presented in Section 3.4.3. Let $\lambda_0 = 15.0$ and $\lambda_1 = 13.0$. Thus,

$$f(\lambda_0) = f(15.0) = \begin{vmatrix} (8 - 15.0) & -2 & -2 \\ -2 & (4 - 15.0) & -2 \\ -2 & -2 & (13 - 15.0) \end{vmatrix} = -90.0 \quad (2.98a)$$

$$f(\lambda_1) = f(13.0) = \begin{vmatrix} (8 - 13.0) & -2 & -2 \\ -2 & (4 - 13.0) & -2 \\ -2 & -2 & (13 - 13.0) \end{vmatrix} = 40.0 \quad (2.98b)$$

The determinants in Eq. (2.98) were evaluated by Gauss elimination, as described in Section 1.3.6. Write the linear relationship between λ and $f(\lambda)$:

$$\frac{f(\lambda_1) - f(\lambda_0)}{\lambda_1 - \lambda_0} = \text{slope} = \frac{f(\lambda_2) - f(\lambda_1)}{\lambda_2 - \lambda_1} \quad (2.99)$$

where $f(\lambda_2) = 0$ is the desired solution. Thus,

$$\text{slope} = \frac{40.0 - (-90.0)}{13.0 - 15.0} = -65.0 \quad (2.100)$$

Solving Eq. (2.99) for λ_2 to give $f(\lambda_2) = 0.0$ gives

$$\lambda_2 = \lambda_1 - \frac{f(\lambda_1)}{\text{slope}} = 13.0 - \frac{40.0}{(-65.0)} = 13.615385 \quad (2.101)$$

The results of the first iteration presented above and subsequent iterations are presented in Table 2.6. The solution is $\lambda = 13.870585$. The solution is quite sensitive to the two initial guesses. These results were obtained on a 13-digit precision computer.

Table 2.6. The Direct Method for a Linear Eigenproblem

k	λ_k (deg)	$f(\lambda_k)$	(slope) $_k$
0	15.000000	-90.000000	
1	13.000000	40.000000	-65.000000
2	13.615385	14.157487	-41.994083
3	13.952515	-4.999194	-56.822743
4	13.864536	0.360200	-60.916914
5	13.870449	0.008098	-59.547441
6	13.870585	-0.000014	-59.647887
7	13.870585	0.000000	

Example 2.6 presents the solution of a linear eigenproblem by the direct method. Nonlinear eigenproblems also can be solved by the direct method, as illustrated in Example 2.7.

Example 2.7. The direct method for a nonlinear eigenproblem.

Consider the nonlinear eigenproblem:

$$x_1 + 0.4x_2 = \sin(\lambda) x_1 \quad (2.102a)$$

$$0.2x_1 + x_2 = \cos(\lambda) x_2 \quad (2.102b)$$

The characteristic determinant corresponding to Eq. (2.102) is

$$f(\lambda) = \det[\mathbf{A} - \mathbf{B}(\lambda)] = \begin{vmatrix} [1 - \sin(\lambda)] & 0.4 \\ 0.2 & [1 - \cos(\lambda)] \end{vmatrix} = 0 \quad (2.103)$$

Let's solve Eq. (2.103) by the secant method. Let $\lambda_0 = 50.0$ deg and $\lambda_1 = 55.0$ deg. Thus,

$$f(\lambda_0) = \begin{vmatrix} [1 - \sin(50)] & 0.4 \\ 0.2 & [1 - \cos(50)] \end{vmatrix} = \begin{vmatrix} 0.233956 & 0.4 \\ 0.2 & 0.357212 \end{vmatrix} = 0.003572 \quad (2.104a)$$

$$f(\lambda_1) = \begin{vmatrix} [1 - \sin(55)] & 0.4 \\ 0.2 & [1 - \cos(55)] \end{vmatrix} = \begin{vmatrix} 0.180848 & 0.4 \\ 0.2 & 0.426424 \end{vmatrix} = -0.002882 \quad (2.104b)$$

Writing the linear relationship between λ and $f(\lambda)$ yields

$$\frac{f(\lambda_1) - f(\lambda_0)}{\lambda_1 - \lambda_0} = \text{slope} = \frac{f(\lambda_2) - f(\lambda_1)}{\lambda_2 - \lambda_1} \quad (2.105)$$

where $f(\lambda_2) = 0.0$ is the desired solution. Thus,

$$\text{slope} = \frac{(-0.002882) - (-0.003572)}{55.0 - 50.0} = -0.001292 \quad (2.106)$$

Solving Eq. (2.105) for λ_2 to give $f(\lambda_2) = 0.0$ yields

$$\lambda_2 = \lambda_1 - \frac{f(\lambda_1)}{\text{slope}} = 55.0 - \frac{(-0.002882)}{(-0.001292)} = 52.767276 \quad (2.107)$$

Table 2.7. The Direct Method for a Nonlinear Eigenproblem

k	λ_k , deg	$f(\lambda_k)$	(Slope) $_k$
0	50.0	0.003572	
1	55.0	-0.002882	-0.001292
2	52.767276	0.000496	-0.001513
3	53.095189	0.000049	-0.001365
4	53.131096	-0.000001	

The results of the first iteration presented above and subsequent iterations are presented in Table 2.7. The solution is $\lambda = 53.131096$ deg. These results were obtained on a 13-digit precision computer and terminated when the change in $f(\lambda)$ between iterations was less than 0.000001.

2.5 THE QR METHOD

The power method presented in Section 2.3 finds individual eigenvalues, as does the direct method presented in Section 2.4. The QR method, on the other hand, finds all of the eigenvalues of a matrix simultaneously. The development of the QR method is presented by Strang (1988). The implementation of the QR method, without proof, is presented in this section.

Triangular matrices have their eigenvalues on the diagonal of the matrix. Consider the upper triangular matrix \mathbf{U} :

$$\mathbf{U} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix} \quad (2.108)$$

The eigenproblem, $(\mathbf{U} - \lambda \mathbf{I})$, is given by

$$(\mathbf{U} - \lambda \mathbf{I}) = \begin{bmatrix} (u_{11} - \lambda) & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & (u_{22} - \lambda) & u_{23} & \cdots & u_{2n} \\ 0 & 0 & (u_{33} - \lambda) & \cdots & u_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \cdots & (u_{nn} - \lambda) \end{bmatrix} \quad (2.109)$$

The characteristic polynomial, $|\mathbf{U} - \lambda \mathbf{I}|$, yields

$$(u_{11} - \lambda)(u_{22} - \lambda)(u_{33} - \lambda) \cdots (u_{nn} - \lambda) = 0 \quad (2.110)$$

The roots of Eq. (2.110) are the eigenvalues of matrix \mathbf{U} . Thus,

$$\lambda_i = u_{i,i} \quad (i = 1, 2, \dots, n) \quad (2.111)$$

The QR method uses similarity transformations to transform matrix \mathbf{A} into triangular form. A similarity transformation is defined as $\mathbf{A}' = \mathbf{M}^{-1} \mathbf{A} \mathbf{M}$. Matrices \mathbf{A} and \mathbf{A}' are said to be similar. The eigenvalues of similar matrices are identical, but the eigenvectors are different.

The *Gram-Schmidt process* starts with matrix \mathbf{A} , whose columns comprise the column vectors $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$, and constructs the matrix \mathbf{Q} , whose columns comprise a set of orthonormal vectors $\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n$. A set of orthonormal vectors is a set of mutually orthogonal unit vectors. The matrix that connects matrix \mathbf{A} to matrix \mathbf{Q} is the upper triangular matrix \mathbf{R} whose elements are the vector products

$$r_{ij} = \mathbf{q}_i^T \mathbf{a}_j \quad (i, j = 1, 2, \dots, n) \quad (2.112)$$

The result is the QR factorization:

$$\boxed{\mathbf{A} = \mathbf{QR}} \quad (2.113)$$

The QR process starts with the Gauss-Schmidt process, Eq. (2.113). That process is then reversed to give

$$\mathbf{A}' = \mathbf{R}\mathbf{Q} \quad (2.114)$$

Matrices \mathbf{A} and \mathbf{A}' can be shown to be similar as follows. Premultiply Eq. (2.113) by \mathbf{Q}^{-1} to obtain

$$\mathbf{Q}^{-1}\mathbf{A} = \mathbf{Q}^{-1}\mathbf{QR} = \mathbf{IR} = \mathbf{R} \quad (2.115)$$

Postmultiply Eq. (2.115) by \mathbf{Q} to obtain

$$\mathbf{Q}^{-1}\mathbf{AQ} = \mathbf{RQ} = \mathbf{A}' \quad (2.116)$$

Equation (2.116) shows that matrices \mathbf{A} and \mathbf{A}' are similar, and thus have the same eigenvalues.

The steps in the Gram-Schmidt process are as follows. Start with the matrix \mathbf{A} expressed as a set of column vectors:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \dots & \dots & \cdots & \dots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_n] \quad (2.117)$$

Assuming that the column vectors \mathbf{a}_i ($i = 1, 2, \dots, n$) are linearly independent, they span the n -dimensional space. Thus, any arbitrary vector can be expressed as a linear combination of the column vectors \mathbf{a}_i ($i = 1, 2, \dots, n$). An orthonormal set of column vectors \mathbf{q}_i ($i = 1, 2, \dots, n$) can be created from the column vectors \mathbf{a}_i ($i = 1, 2, \dots, n$) by the following steps.

Choose \mathbf{q}_1 to have the direction of \mathbf{a}_1 . Then normalize \mathbf{a}_1 to obtain \mathbf{q}_1 :

$$\mathbf{q}_1 = \frac{\mathbf{a}_1}{\|\mathbf{a}_1\|} \quad (2.118a)$$

where $\|\mathbf{a}_1\|$ denotes the magnitude of \mathbf{a}_1 :

$$\|\mathbf{a}_1\| = [a_{11}^2 + a_{12}^2 + \cdots + a_{n1}^2]^{1/2} \quad (2.118b)$$

To determine \mathbf{q}_2 , first subtract the component of \mathbf{a}_2 in the direction of \mathbf{q}_1 to determine vector \mathbf{a}'_2 , which is normal to \mathbf{q}_1 . Thus,

$$\mathbf{a}'_2 = \mathbf{a}_2 - (\mathbf{q}_1^T \mathbf{a}_2) \mathbf{q}_1 \quad (2.119a)$$

Choose \mathbf{q}_2 to have the direction of \mathbf{a}'_2 . Then normalize \mathbf{a}'_2 to obtain \mathbf{q}_2 :

$$\mathbf{q}_2 = \frac{\mathbf{a}'_2}{\|\mathbf{a}'_2\|} \quad (2.119b)$$

This process continues until a complete set of n orthonormal unit vectors is obtained. Let's evaluate one more orthonormal vector \mathbf{q}_3 to illustrate the process. To determine \mathbf{q}_3 , first subtract the components of \mathbf{a}_3 in the directions of \mathbf{q}_1 and \mathbf{q}_2 . Thus,

$$\mathbf{a}'_3 = \mathbf{a}_3 - (\mathbf{q}_1^T \mathbf{a}_3) \mathbf{q}_1 - (\mathbf{q}_2^T \mathbf{a}_3) \mathbf{q}_2 \quad (2.120a)$$

Choose \mathbf{q}_3 to have the direction of \mathbf{a}'_3 . Then normalize \mathbf{a}'_3 to obtain \mathbf{q}_3 :

$$\mathbf{q}_3 = \frac{\mathbf{a}'_3}{\|\mathbf{a}'_3\|} \quad (2.120b)$$

The general expression for \mathbf{a}'_i is

$$\mathbf{a}'_i = \mathbf{a}_i - \sum_{k=1}^{i-1} (\mathbf{q}_k^T \mathbf{a}_i) \mathbf{q}_k \quad (i = 2, 3, \dots, n) \quad (2.121)$$

and the general expression for \mathbf{q}_i is

$$\mathbf{q}_i = \frac{\mathbf{a}'_i}{\|\mathbf{a}'_i\|} \quad (i = 1, 2, \dots, n) \quad (2.122)$$

The matrix \mathbf{Q} is composed of the column vectors \mathbf{q}_i ($i = 1, 2, \dots, n$). Thus,

$$\mathbf{Q} = [\mathbf{q}_1 \quad \mathbf{q}_2 \quad \cdots \quad \mathbf{q}_n] \quad (2.123)$$

The upper triangular matrix \mathbf{R} is assembled from the elements computed in the evaluation of \mathbf{Q} . The diagonal elements of \mathbf{R} are the magnitudes of the \mathbf{a}'_i vectors:

$$r_{i,i} = \|\mathbf{a}'_i\| \quad (i = 1, 2, \dots, n) \quad (2.124)$$

The off-diagonal elements of \mathbf{R} are the components of the \mathbf{a}_i vectors which are subtracted from the \mathbf{a}_i vectors in the evaluation of the \mathbf{a}'_i vectors. Thus,

$$r_{i,j} = \mathbf{q}_i^T \mathbf{a}_j \quad (i = 1, 2, \dots, n, j = i+1, \dots, n) \quad (2.125)$$

The values of $r_{i,i}$ and $r_{i,j}$ are calculated during the evaluation of the orthonormal unit vectors \mathbf{q}_i . Thus, \mathbf{R} is simply assembled from already calculated values. Thus,

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \cdots & r_{nn} \end{bmatrix} \quad (2.126)$$

The first step in the QR process is to set $\mathbf{A}^{(0)} = \mathbf{A}$ and factor $\mathbf{A}^{(0)}$ by the Gram-Schmidt process into $\mathbf{Q}^{(0)}$ and $\mathbf{R}^{(0)}$. The next step is to reverse the factors $\mathbf{Q}^{(0)}$ and $\mathbf{R}^{(0)}$ to obtain

$$\mathbf{A}^{(1)} = \mathbf{R}^{(0)} \mathbf{Q}^{(0)} \quad (2.127)$$

$\mathbf{A}^{(1)}$ is similar to \mathbf{A} , so the eigenvalues are preserved. $\mathbf{A}^{(1)}$ is factored by the Gram-Schmidt process to obtain $\mathbf{Q}^{(1)}$ and $\mathbf{R}^{(1)}$, and the factors are reversed to obtain $\mathbf{A}^{(2)}$. Thus,

$$\mathbf{A}^{(2)} = \mathbf{R}^{(1)} \mathbf{Q}^{(1)} \quad (2.128)$$

The process is continued to determine $\mathbf{A}^{(3)}$, $\mathbf{A}^{(4)}$, ..., $\mathbf{A}^{(n)}$. When $\mathbf{A}^{(n)}$ approaches triangular form, within some tolerance, the eigenvalues of \mathbf{A} are the diagonal elements. The process is as follows:

$$\mathbf{A}^{(k)} = \mathbf{Q}^{(k)} \mathbf{R}^{(k)} \quad (2.129)$$

$$\mathbf{A}^{(k+1)} = \mathbf{R}^{(k)} \mathbf{Q}^{(k)} \quad (2.130)$$

Equations (2.129) and (2.130) are the basic QR algorithm. Although it generally converges, it can be fairly slow. Two modifications are usually employed to increase its speed:

1. Preprocessing matrix \mathbf{A} into a more nearly triangular form
2. Shifting the eigenvalues as the process proceeds

With these modifications, the QR algorithm is generally the preferred method for solving eigenproblems.

Example 2.8. The basic QR method.

Let's apply the QR method to find all the eigenvalues of matrix \mathbf{A} given by Eq. (2.11) simultaneously. Recall:

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \quad (2.131)$$

The column vectors associated with matrix \mathbf{A} , $\mathbf{A} = [\mathbf{a}_1 \ \mathbf{a}_2 \ \mathbf{a}_3]$, are

$$\mathbf{a}_1 = \begin{bmatrix} 8 \\ -2 \\ -2 \end{bmatrix} \quad \mathbf{a}_2 = \begin{bmatrix} -2 \\ 4 \\ -2 \end{bmatrix} \quad \mathbf{a}_3 = \begin{bmatrix} -2 \\ -2 \\ 13 \end{bmatrix} \quad (2.132)$$

First let's solve for \mathbf{q}_1 . Let \mathbf{q}_1 have the direction of \mathbf{a}_1 , and divide by the magnitude of \mathbf{a}_1 . Thus,

$$\|\mathbf{a}_1\| = [8^2 + (-2)^2 + (-2)^2]^{1/2} = 8.485281 \quad (2.133)$$

Solving for $\mathbf{q}_1 = \mathbf{a}_1 / \|\mathbf{a}_1\|$ gives

$$\mathbf{q}_1^T = [0.942809 \quad -0.235702 \quad -0.235702] \quad (2.134)$$

Next let's solve for \mathbf{q}_2 . First subtract the component of \mathbf{a}_2 in the direction of \mathbf{q}_1 :

$$\mathbf{a}'_2 = \mathbf{a}_2 - (\mathbf{q}_1^T \mathbf{a}_2) \mathbf{q}_1 \quad (2.135a)$$

$$\mathbf{a}'_2 = \begin{bmatrix} -2 \\ 4 \\ -2 \end{bmatrix} - [0.942809 \quad -0.235702 \quad -0.235702] \begin{bmatrix} -2 \\ 4 \\ -2 \end{bmatrix} \begin{bmatrix} 0.942809 \\ -0.235702 \\ -0.235702 \end{bmatrix} \quad (2.135b)$$

Performing the calculations gives $\mathbf{q}_1^T \mathbf{a}_2 = -2.357023$ and

$$\mathbf{a}'_2 = \begin{bmatrix} -2 & -(-2.222222) \\ 4 & -(0.555555) \\ -2 & -(0.555555) \end{bmatrix} = \begin{bmatrix} 0.222222 \\ 3.444444 \\ -2.555557 \end{bmatrix} \quad (2.136)$$

The magnitude of \mathbf{a}'_2 is $\|\mathbf{a}'_2\| = 4.294700$. Thus, $\mathbf{q}_2 = \mathbf{a}'_2 / \|\mathbf{a}'_2\|$ gives

$$\mathbf{q}_2 = [0.051743 \quad 0.802022 \quad -0.595049] \quad (2.137)$$

Finally let's solve for \mathbf{q}_3 . First subtract the components of \mathbf{a}_3 in the directions of \mathbf{q}_1 and \mathbf{q}_2 :

$$\mathbf{a}'_3 = \mathbf{a}_3 - (\mathbf{q}_1^T \mathbf{a}_3) \mathbf{q}_1 - (\mathbf{q}_2^T \mathbf{a}_3) \mathbf{q}_2 \quad (2.138)$$

$$\begin{aligned} \mathbf{a}'_3 = & \begin{bmatrix} -2 \\ -2 \\ 13 \end{bmatrix} - [0.942809 \quad -0.235702 \quad -0.235702] \begin{bmatrix} -2 \\ -2 \\ 13 \end{bmatrix} \begin{bmatrix} 0.942809 \\ -0.235702 \\ -0.235702 \end{bmatrix} \\ & -[0.051743 \quad 0.802022 \quad -0.595049] \begin{bmatrix} -2 \\ -2 \\ 13 \end{bmatrix} \begin{bmatrix} 0.051743 \\ 0.802022 \\ -0.595049 \end{bmatrix} \end{aligned} \quad (2.139)$$

Performing the calculations gives $\mathbf{q}_1^T \mathbf{a}_3 = -4.478343$, $\mathbf{q}_2^T \mathbf{a}_3 = -9.443165$, and

$$\mathbf{a}'_3 = \begin{bmatrix} -2 & -(-4.222222) & -(-0.488618) \\ -2 & -(1.055554) & -(-7.573626) \\ 13 & -(1.055554) & -(5.619146) \end{bmatrix} = \begin{bmatrix} 2.710840 \\ 4.518072 \\ 6.325300 \end{bmatrix} \quad (2.140)$$

The magnitude of \mathbf{a}'_3 is $\|\mathbf{a}'_3\| = 8.232319$. Thus, $\mathbf{q}_3 = \mathbf{a}'_3 / \|\mathbf{a}'_3\|$ gives

$$\mathbf{q}'_3 = [0.329293 \quad 0.548821 \quad 0.768350] \quad (2.141)$$

In summary, matrix $\mathbf{Q}^{(0)} = [\mathbf{q}_1 \quad \mathbf{q}_2 \quad \mathbf{q}_3]$ is given by

$$\mathbf{Q}^{(0)} = \begin{bmatrix} 0.942809 & 0.051743 & 0.329293 \\ -0.235702 & 0.802022 & 0.548821 \\ -0.235702 & -0.595049 & 0.768350 \end{bmatrix} \quad (2.142)$$

Matrix $\mathbf{R}^{(0)}$ is assembled from the elements computed in the calculation of matrix $\mathbf{Q}^{(0)}$. Thus, $r_{11} = \|\mathbf{a}'_1\| = 8.485281$, $r_{22} = \|\mathbf{a}'_2\| = 4.294700$, and $r_{33} = \|\mathbf{a}'_3\| = 8.232319$. The off-diagonal elements are $r_{12} = \mathbf{q}_1^T \mathbf{a}_2 = -2.357023$, $r_{13} = \mathbf{q}_1^T \mathbf{a}_3 = -4.478343$, and $r_{23} = \mathbf{q}_2^T \mathbf{a}_3 = -9.443165$. Thus, matrix $\mathbf{R}^{(0)}$ is given by

$$\mathbf{R}^{(0)} = \begin{bmatrix} 8.485281 & -2.357023 & -4.478343 \\ 0.000000 & 4.294700 & -9.443165 \\ 0.000000 & 0.000000 & 8.232819 \end{bmatrix} \quad (2.143)$$

It can be shown by matrix multiplication that $\mathbf{A}^{(0)} = \mathbf{Q}^{(0)} \mathbf{R}^{(0)}$.

Table 2.8. The Basic QR Method

k	λ_1	λ_2	λ_3
1	9.611111	9.063588	6.325301
2	10.743882	11.543169	2.712949
3	11.974170	10.508712	2.517118
4	12.929724	9.560916	2.509360
...
19	13.870584	8.620435	2.508981
20	13.870585	8.620434	2.508981

The next step is to evaluate matrix $\mathbf{A}^{(1)} = \mathbf{R}^{(0)}\mathbf{Q}^{(0)}$. Thus,

$$\mathbf{A}^{(1)} = \begin{bmatrix} 8.485281 & -2.357023 & -4.478343 \\ 0.000000 & 4.294700 & -9.443165 \\ 0.000000 & 0.000000 & 8.232819 \end{bmatrix} \times \begin{bmatrix} 0.942809 & 0.051743 & 0.329293 \\ -0.235702 & 0.802022 & 0.548821 \\ -0.235702 & -0.595049 & 0.768350 \end{bmatrix} \quad (2.144)$$

$$\mathbf{A}^{(1)} = \begin{bmatrix} 9.611111 & 1.213505 & -1.940376 \\ 1.213505 & 9.063588 & -4.898631 \\ -1.940376 & -4.898631 & 6.325301 \end{bmatrix} \quad (2.145)$$

The diagonal elements in matrix $\mathbf{A}^{(1)}$ are the first approximation to the eigenvalues of matrix \mathbf{A} . The results of the first iteration presented above and subsequent iterations are presented in Table 2.8. The final values of matrices \mathbf{Q} , \mathbf{R} , and \mathbf{A} are given below:

$$\mathbf{Q}^{(19)} = \begin{bmatrix} 1.000000 & -0.000141 & 0.000000 \\ 0.000141 & 1.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 \end{bmatrix} \quad (2.146)$$

$$\mathbf{R}^{(19)} = \begin{bmatrix} 13.870585 & 0.003171 & 0.000000 \\ 0.000000 & 8.620434 & 0.000000 \\ 0.000000 & 0.000000 & 2.508981 \end{bmatrix} \quad (2.147)$$

$$\mathbf{A}^{(20)} = \begin{bmatrix} 13.870585 & 0.001215 & 0.000000 \\ 0.001215 & 8.620434 & 0.000000 \\ 0.000000 & 0.000000 & 2.508981 \end{bmatrix} \quad (2.148)$$

The final values agree well with the values obtained by the power method and summarized at the end of Section 2.3. The QR method does not yield the corresponding eigenvectors. The eigenvector corresponding to each eigenvalue can be found by the inverse shifted power method presented in Section 2.6.

2.6 EIGENVECTORS

Some methods for solving eigenproblems, such as the power method, yield both the eigenvalues and the corresponding eigenvectors. Other methods, such as the direct method and the QR method, yield only the eigenvalues. In these cases, the corresponding eigenvectors can be evaluated by shifting the matrix by the eigenvalues and applying the inverse power method one time.

Example 2.9. Eigenvectors.

Let's apply this technique to evaluate the eigenvector \mathbf{x}_1 corresponding to the largest eigenvalue of matrix \mathbf{A} , $\lambda_1 = 13.870584$, which was obtained in Example 2.1. From that example,

$$\mathbf{A} = \begin{bmatrix} 8 & -2 & -2 \\ -2 & 4 & -2 \\ -2 & -2 & 13 \end{bmatrix} \quad (2.149)$$

Shifting matrix \mathbf{A} by $\lambda = 13.870584$, gives

$$\mathbf{A}_{\text{shifted}} = (\mathbf{A} - s\mathbf{I}) = \begin{bmatrix} -5.870584 & -2.000000 & -2.000000 \\ -2.000000 & -9.870584 & -2.000000 \\ -2.000000 & -2.000000 & -0.870584 \end{bmatrix} \quad (2.150)$$

Applying the Doolittle LU method to $\mathbf{A}_{\text{shifted}}$ yields \mathbf{L} and \mathbf{U} :

$$\mathbf{L} = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 \\ 0.340682 & 1.000000 & 0.000000 \\ 0.340682 & 0.143498 & 1.000000 \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} -5.870584 & -2.000000 & -2.000000 \\ 0.000000 & -9.189221 & -1.318637 \\ 0.000000 & 0.000000 & 0.000001 \end{bmatrix} \quad (2.151)$$

Let the initial guess for $\mathbf{x}^{(0)T} = [1.0 \quad 1.0 \quad 1.0]$. Solve for \mathbf{x}' by forward substitution using $\mathbf{L}\mathbf{x}' = \mathbf{x}$.

$$\begin{bmatrix} 1.000000 & 0.000000 & 0.000000 \\ 0.340682 & 1.000000 & 0.000000 \\ 0.340682 & 0.143498 & 1.000000 \end{bmatrix} \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} \rightarrow \begin{aligned} x'_1 &= 1.000000 \\ x'_2 &= 0.659318 \\ x'_3 &= 0.564707 \end{aligned} \quad (2.152)$$

Solve for \mathbf{y} by back substitution using $\mathbf{U}\mathbf{y} = \mathbf{x}'$.

$$\begin{bmatrix} -5.870584 & -2.000000 & -2.000000 \\ 0.000000 & -9.189221 & -1.318637 \\ 0.000000 & 0.000000 & 0.000001 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1.000000 \\ 0.659318 \\ 0.564707 \end{bmatrix} \quad (2.153)$$

The solution, including scaling the first component to unity, is

$$\mathbf{y} = \begin{bmatrix} -0.132662 \times 10^6 \\ -0.652404 \times 10^6 \\ 0.454642 \times 10^6 \end{bmatrix} \rightarrow 0.454642 \times 10^6 \begin{bmatrix} -0.291794 \\ -0.143498 \\ 1.000000 \end{bmatrix} \quad (2.154)$$

Thus, $\mathbf{x}^T = [-0.291794 \quad -0.143498 \quad 1.000000]$, which is identical to the value obtained by the direct power method in Example 2.1.

2.7 OTHER METHODS

The power method, including its variations, and the direct method are very inefficient when all the eigenvalues of a large matrix are desired. Several other methods are available in such cases. Most of these methods are based on a two-step procedure. In the first step, the original matrix is transformed to a simpler form that has the same eigenvalues as the original matrix. In the second step, iterative procedures are used to determine these eigenvalues. The best general purpose method is generally considered to be the *QR method*, which is presented in Section 2.5.

Most of the more powerful methods apply to special types of matrices. Many of them apply to symmetric matrices. Generally speaking, the original matrix is transformed into a simpler form that has the same eigenvalues. Iterative methods are then employed to evaluate the eigenvalues. More information on the subject can be found in Fadeev and Fadeeva (1963), Householder (1964), Wilkinson (1965), Steward (1973), Ralston and Rabinowitz (1978), and Press, Flannery, Teukolsky, and Vetterling (1989). Numerous compute programs for solving eigenproblems can be found in the IMSL (International Mathematical and Statistics Library) library and in the EISPACK program (Argonne National Laboratory). See Rice (1983) and Smith et al. (1976) for a discussion of these programs.

The Jacobi method transforms a symmetric matrix into a diagonal matrix. The off-diagonal elements are eliminated in a systematic manner. However, elimination of subsequent off-diagonal elements creates nonzero values in previously eliminated elements. Consequently, the transformation approaches a diagonal matrix iteratively. The Given method and the Householder method reduce a symmetric matrix to a tridiagonal matrix in a direct rather than an iterative manner. Consequently, they are more efficient than the Jacobi method. The resulting tridiagonal matrix can be expanded, and the corresponding characteristic equation can be solved for the eigenvalues by iterative techniques.

For more general matrices, the QR method is recommended. Due to its robustness, the QR method is generally the method of choice. See Wilkinson (1965) and Strang (1988) for a discussion of the QR method. The Householder method can be applied to nonsymmetrical matrices to reduce them to Hessenberg matrices, whose eigenvalues can then be found by the QR method.

Finally, deflation techniques can be employed for symmetric matrices. After the largest eigenvalue λ_1 of matrix \mathbf{A} is found, for example, by the power method, a new matrix \mathbf{B} is formed whose eigenvalues are the same as the eigenvalues of matrix \mathbf{A} , except that the largest eigenvalue λ_1 is replaced by zero in matrix \mathbf{B} . The power method can then be applied to matrix \mathbf{B} to determine its largest eigenvalue, which is the second largest eigenvalue λ_2 of matrix \mathbf{A} . In principle, deflation can be applied repetitively to find all the eigenvalues of matrix \mathbf{A} . However, round-off errors generally pollute the results after a few deflations. The results obtained by deflation can be used to shift matrix \mathbf{A} by the approximate eigenvalues, which are then solved for by the shifted inverse power method presented in Section 2.3.4 to find more accurate values.

2.8 PROGRAMS

Two FORTRAN subroutines for solving eigenproblems are presented in this section:

1. The power method
2. The inverse power method

The basic computational algorithms are presented as completely self-contained subroutines suitable for use in other programs. Input data and output statements are contained in a main (or driver) program written specifically to illustrate the use of each subroutine.

2.8.1. The Power Method

The *direct power method* evaluates the largest (in magnitude) eigenvalue of a matrix. The general algorithm for the power method is given by Eq. (2.39):

$$\mathbf{A}\mathbf{x}^{(k)} = \mathbf{y}^{(k+1)} = \lambda^{(k+1)}\mathbf{x}^{(k+1)} \quad (2.155)$$

A FORTRAN subroutine, *subroutine power*, for implementing the direct power method is presented below. *Subroutine power* performs the matrix multiplication, $\mathbf{Ax} = \mathbf{y}$, factors out the approximate eigenvalue λ to obtain the approximate eigenvector \mathbf{x} , checks for convergence, and returns or continues. After *iter* iterations, an error message is printed out and the iteration is terminated. *Program main* defines the data set and prints it, calls *subroutine power* to implement the solution, and prints the solution.

Program 2.1. The direct power method program.

```

program main
c   main program to illustrate eigenproblem solvers
c   ndim  array dimension, ndim = 6 in this example
c   n      number of equations, n
c   a      coefficient matrix, A(i,j)
c   x      eigenvector, x(i)
c   y      intermediate vector, y(i)
c   norm  specifies unity component of eigenvector
c   iter  number of iterations allowed
c   tol   convergence tolerance
c   shift amount by which eigenvalue is shifted
c   iw    intermediate results output flag: 0 no, 1 yes
dimension a(6,6),x(6),y(6)
data ndim,n,norm,iter,tol,shift,iw / 6,3,3,50,1.e-6,0.0,1/
data ndim,n,norm,iter,tol,shift,iw/6,3,3,50,1.e-6,13.870584,2/
data (a(i,1),i=1,3) / 8.0, -2.0, -2.0 /
data (a(i,2),i=1,3) / -2.0, 4.0, -2.0 /
data (a(i,3),i=1,3) / -2.0, -2.0, 13.0 /
data (x(i),i=1,3) / 1.0, 1.0, 1.0 /
write (6,1000)
do i=1,n
    write (6,1010) i,(a(i,j),j=1,n),x(i)
end do

```

```

if (shift.gt.0.0) then
    write (6,1005) shift
    do i=1,n
        a(i,i)=a(i,i)-shift
        write (6,1010) i,(a(i,j),j=1,n)
    end do
end if
call power (ndim,n,a,x,y,norm,iter,tol,shift,iw,k,ev2)
write (6,1020)
write (6,1010) k,ev2,(x(i),i=1,n)
stop
1000 format (' The power method'// ' ' A and x(0)'// ')
1005 format (' '// A shifted by shift = ',f10.6'// ')
1010 format (1x,i3,6f12.6)
1020 format (' '// k      lambda and eigenvector components'// ')
end

subroutine power (ndim,n,a,x,y,norm,iter,tol,shift,iw,k,ev2)
c   the direct power method
dimension a(ndim,ndim),x(ndim),y(ndim)
ev1=0.0
if (iw.eq.1) write (6,1000)
if (iw.eq.1) write (6,1010) k,ev1,(x(i),i=1,n)
do k=1,iter
c   calculate y(i)
    do i=1,n
        y(i)=0.0
        do j=1,n
            y(i)=y(i)+a(i,j)*x(j)
        end do
    end do
c   calculate lambda and x(i)
    ev2=y(norm)
    if (abs(ev2).le.1.0e-3) then
        write (6,1020) ev2
        return
    else
        do i=1,n
            x(i)=y(i)/ev2
        end do
    end if
    if (iw.eq.1) write (6,1010) k,ev2,(x(i),i=1,n)
c   check for convergence
    if (abs(ev2-ev1).le.tol) then
        if (shift.ne.0.0) then
            ev1=ev2
            ev2=ev2+shift
            write (6,1040) ev1,ev2
        end if
        return
    else
        ev1=ev2
    end if
end

```

```

    end if
end do
write (6,1030)
return
1000 format (' '' k      lambda and eigenvector components'' ')
1010 format (1x,i3,6f12.6)
1020 format (' '' lambda = ',e10.2,' approaching zero, stop')
1030 format (' '' Iterations did not converge, stop')
1040 format (' '' lambda shifted =',f12.6,' and lambda =',f12.6)
end

```

The data set used to illustrate the use of *subroutine power* is taken from Example 2.1. The output generated by the power method program is presented below.

Output 2.1. Solution by the direct power method.

The power method

A and x(0)

1	8.000000	-2.000000	-2.000000	1.000000
2	-2.000000	4.000000	-2.000000	1.000000
3	-2.000000	-2.000000	13.000000	1.000000

k lambda and eigenvector components

0	0.000000	1.000000	1.000000	1.000000
1	9.000000	0.444444	0.000000	1.000000
2	12.111111	0.128440	-0.238532	1.000000
3	13.220183	-0.037474	-0.242887	1.000000
4	13.560722	-0.133770	-0.213602	1.000000
5	13.694744	-0.192991	-0.188895	1.000000
..
29	13.870583	-0.291793	-0.143499	1.000000
30	13.870584	-0.291794	-0.143499	1.000000

k lambda and eigenvector components

30	13.870584	-0.291794	-0.143499	1.000000
----	-----------	-----------	-----------	----------

Subroutine power also implements the *shifted direct power method*. If the input variable, *shift*, is nonzero, matrix *A* is shifted by the value of *shift* before the direct power method is implemented. This implements the shifted direct power method. Example 2.3 illustrating the shifted direct power method can be solved by *subroutine power* simply by defining *norm* = 2 and *shift* = 13.870584 in the *data* statement. The *data* statement for this additional case is included in *program main* as a *comment* statement.

2.8.2. The Inverse Power Method

The *inverse power method* evaluates the largest (in magnitude) eigenvalue of the inverse matrix, \mathbf{A}^{-1} . The general equation for the inverse power method is given by

$$\mathbf{A}^{-1}\mathbf{x} = \lambda_{\text{inverse}}\mathbf{x} \quad (2.156)$$

This can be accomplished by evaluating \mathbf{A}^{-1} by Gauss-Jordan elimination applied to the identity matrix \mathbf{I} or by using the Doolittle LU factorization approach described in Section 2.3.3. Since a subroutine for Doolittle LU factorization is presented in Section 1.8.2, that approach is taken here. The general algorithm for the inverse power method based on the LU factorization approach is given by Eqs. (2.62) to (2.64):

$$\mathbf{L}\mathbf{x}' = \mathbf{x}^{(k)} \quad (2.157a)$$

$$\mathbf{U}\mathbf{y}^{(k+1)} = \mathbf{x}' \quad (2.157b)$$

$$\mathbf{y}^{(k+1)} = \lambda_{\text{inverse}}^{(k+1)}\mathbf{x}^{(k+1)} \quad (2.157c)$$

A FORTRAN subroutine, *subroutine invpower*, for implementing the inverse power method is presented below. *Program main* defines the data set and prints it, calls *subroutine invpower* to implement the inverse power method, and prints the solution. *Subroutine invpower* calls *subroutine lufactor* and *subroutine solve* from Section 1.8.2 to evaluate \mathbf{L} and \mathbf{U} . This is indicated in *subroutine invpower* by including the subroutine declaration statements. The subroutines themselves must be included when *subroutine invpower* is to be executed. *Subroutine invpower* then evaluates \mathbf{x}' , $\mathbf{y}^{(k+1)}$, $\lambda_{\text{inverse}}^{(k+1)}$, and $\mathbf{x}^{(k+1)}$. Convergence of λ is checked, and the solution continues or returns. After *iter* iterations, an error message is printed and the solution is terminated. *Program main* in this section contains only the statements which are different from the statements in *program main* in Section 2.8.1.

Program 2.2. The inverse power method program.

```

program main
c      main program to illustrate eigenproblem solvers
c      xp      intermediate solution vector
      dimension a(6,6),x(6),xp(6),y(6)
      data ndim,n,norm,iter,tol,shift,iw / 6,3,1,50,1.e-6,0.0,1 /
      data ndim,n,norm,iter,tol,shift,iw / 6,3,1,50,1.e-6,10.0,1 /
      data ndim,n,norm,iter,tol,shift,iw/6,3,3,50,1.e-6,13.694744,1/
      data ndim,n,norm,iter,tol,shift,iw/6,3,3,1,1.e-6,13.870584,1/
      data (x(i),i=1,3) / 1.0, 1.0, 1.0 /
      data (x(i),i=1,3) / -0.192991, -0.188895, 1.0 /
      call invpower (ndim,n,a,x,xp,y,norm,iter,tol,iw,shift,k,ev2)
1000 format (' The inverse power method'// ' A and x(0)'// ')
      end

      subroutine invpower (ndim,n,a,x,xp,y,norm,iter,tol,iw,shift,k,
1 ev2)
c      the inverse power method.
      dimension a(ndim,ndim),x(ndim),xp(ndim),y(ndim)

```

```

c      perform the LU factorization
call lufactor (ndim,n,a)
if (iw.eq.1) then
    write (6,1000)
    do i=1,n
        write (6,1010) i,(a(i,j),j=1,n)
    end do
end if
if (iw.eq.1) write (6,1005) (x(i),i=1,n)
c      iteration loop
do k=1,iter
    call solve (ndim,n,a,x,xp,y)
    ev2=y(norm)
    if (abs(ev2).le.1.0e-3) then
        write (6,1020) ev2
        return
    else
        do i=1,n
            x(i)=y(i)/ev2
        end do
    end if
    if (iw.eq.1) then
        write (6,1010) k,(xp(i),i=1,n)
        write (6,1015) (y(i),i=1,n)
        write (6,1015) (x(i),i=1,n),ev2
    end if
c      check for convergence
    if (abs(ev2-ev1).le.tol) then
        ev1=ev2
        ev2=1.0/ev2
        if (iw.eq.1) write (6,1040) ev1,ev2
        if (shift.ne.0.0) then
            ev1=ev2
            ev2=ev2+shift
            if (iw.eq.1) write (6,1050) ev1,ev2
        end if
        return
    else
        ev1=ev2
    end if
end do
if (iter.gt.1) write (6,1030)
return
1000 format (' '' L and U matrices stored in matrix A'' ')
1005 format (' '' row 1: k, xprime; row 2: y; row 3: x, ev2'
  1 '' /4x,6f12.6)
1010 format (1x,i3,6f12.6)
1015 format (4x,6f12.6)
1020 format (' '' ev2 = ',e10.2,' is approaching zero, stop')
1030 format (' '' Iterations did not converge, stop')
1040 format (' '' lambda inverse =',f12.6,' and lambda ='f12.6)

```

```

1050 format (' // lambda shifted =',f12.6,' and lambda =',f12.6)
end

subroutine lufactor (ndim,n,a)
c      implements LU factorization and stores L and U in A
end

subroutine solve (ndim,n,a,b,bp,x)
c      process b to b' and b' to x
end

```

The data set used to illustrate *subroutine invpower* is taken from Example 2.2. The output generated by the inverse power method program is presented below.

Output 2.2. Solution by the inverse power method.

The inverse power method

A and x(0)

1	8.000000	-2.000000	-2.000000	1.000000
2	-2.000000	4.000000	-2.000000	1.000000
3	-2.000000	-2.000000	13.000000	1.000000

L and U matrices stored in matrix A

1	8.000000	-2.000000	-2.000000
2	-0.250000	3.500000	-2.500000
3	-0.250000	-0.714286	10.714286

row 1: k, xprime; row 2: y; row 3: x, ev2

1	1.000000	1.000000	1.000000	
1	1.000000	1.250000	2.142857	
	0.300000	0.500000	0.200000	
	1.000000	1.666667	0.666667	0.300000
2	1.000000	1.916667	2.285714	
	0.353333	0.700000	0.213333	
	1.000000	1.981132	0.603774	0.353333
3	1.000000	2.231132	2.447439	
	0.382264	0.800629	0.228428	
	1.000000	2.094439	0.597565	0.382264
..
12	1.000000	2.395794	2.560994	
	0.398568	0.855246	0.239026	
	1.000000	2.145796	0.599712	0.398568

lambda inverse = 0.398568 and lambda = 2.508983

k lambda and eigenvector components

12	2.508983	1.000000	2.145796	0.599712
----	----------	----------	----------	----------

Subroutine invpower also implements the *shifted inverse power method*. If the input variable, *shift*, is nonzero, matrix A is shifted by the value of *shift* before the inverse power method is implemented. This implements the shifted inverse power method. Example 2.4 illustrating the evaluation of an intermediate eigenvalue by the shifted inverse power method can be solved by *subroutine invpower* simply by defining *shift* = 10.0 in the *data* statement. Example 2.5 illustrating shifting eigenvalues to accelerate convergence by the shifted inverse power method can be solved by *subroutine invpower* by defining *norm* = 3 and *shift* = 13.694744 in the *data* statement and defining *x(i)* = -0.192991, -0.188895, 1.0. Example 2.9 illustrating the evaluation of the eigenvector corresponding to a known eigenvalue can be solved by *subroutine invpower* simply by defining *shift* = 13.870584 and *iter* = 1 in the *data* statement. The *data* statements for these additional cases are included in *program main* as *comment* statements.

2.8.3. Packages for Eigenproblems

Numerous libraries and software packages are available for solving eigenproblems. Many workstations and mainframe computers have such libraries attached to their operating systems. If not, libraries such as EISPACK can be added to the operating systems.

Many commercial software packages contain eigenproblem solvers. Some of the more prominent packages are Matlab and Mathcad. More sophisticated packages, such as ISML, Mathematica, Macsyma, and Maple, also contain eigenproblem solvers. Finally, the book *Numerical Recipes* [Press et al. (1989)] contains subroutines and advice for solving eigenproblems.

2.9 SUMMARY

Some general guidelines for solving eigenproblems are summarized below.

- When only the largest and/or smallest eigenvalue of a matrix is required, the power method can be employed.
- Although it is rather inefficient, the power method can be used to solve for intermediate eigenvalues.
- The direct method is not a good method for solving linear eigenproblems. However, it can be used for solving nonlinear eigenproblems.
- For serious eigenproblems, the QR method is recommended.
- Eigenvectors corresponding to a known eigenvalue can be determined by one application of the shifted inverse power method.

After studying Chapter 2, you should be able to:

1. Explain the physical significance of an eigenproblem.
2. Explain the mathematical characteristics of an eigenproblem.
3. Explain the basis of the power method.
4. Solve for the largest (in absolute value) eigenvalue of a matrix by the power method.
5. Solve for the smallest (in absolute value) eigenvalue of a matrix by the inverse power method.
6. Solve for the opposite extreme eigenvalue of a matrix by shifting the eigenvalues of the matrix by the largest (in absolute value) eigenvalue and applying

the inverse power method to the shifted matrix. This procedure yields either the smallest (in absolute value) eigenvalue or the largest (in absolute value) eigenvalue of opposite sign.

7. Solve for an intermediate eigenvalue of a matrix by shifting the eigenvalues of the matrix by an estimate of the intermediate eigenvalue and applying the inverse power method to the shifted matrix.
8. Accelerate the convergence of an eigenproblem by shifting the eigenvalues of the matrix by an approximate value of the eigenvalue obtained by another method, such as the direct power method, and applying the inverse power method to the shifted matrix.
9. Solve for the eigenvalues of a linear or nonlinear eigenproblem by the direct method.
10. Solve for the eigenvalues of a matrix by the QR method.
11. Solve for the eigenvector corresponding to a known eigenvalue of a matrix by applying the inverse power method one time.

EXERCISE PROBLEMS

Consider the linear eigenproblem, $\mathbf{Ax} = \lambda\mathbf{x}$, for the matrices given below. Solve the problems presented below for the specified matrices. Carry at least six figures after the decimal place. Iterate until the values of λ change by less than three digits after the decimal place. Begin all problems with $\mathbf{x}^{(0)T} = [1.0 \quad 1.0 \quad \dots \quad 1.0]$ unless otherwise specified. Show all the results for the first three iterations. Tabulate the results of subsequent iterations. Several of these problems require a large number of iterations.

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 3 & 2 \\ 3 & 4 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 1 \\ 1 & 1 & 3 \end{bmatrix} \quad \mathbf{E} = \begin{bmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{F} = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 2 & 1 & 1 & 1 \\ 3 & 2 & 1 & 2 \\ 2 & 1 & 1 & 4 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 2 & 1 & 1 & 1 \\ 3 & 2 & 1 & 2 \\ 2 & 1 & 1 & 4 \end{bmatrix}$$

2.2 Basic Characteristics of Eigenproblems

1. Solve for the eigenvalues of (a) matrix \mathbf{A} , (b) matrix \mathbf{B} , and (c) matrix \mathbf{C} by expanding the determinant of $(\mathbf{A} - \lambda\mathbf{I})$ and solving the characteristic equation by the quadratic formula. Solve for the corresponding eigenvectors by substituting the eigenvalues into the equation $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0$ and solving for \mathbf{x} . Let the first component of \mathbf{x} be unity.
2. Solve for the eigenvalues of (a) matrix \mathbf{D} , (b) matrix \mathbf{E} , and (c) matrix \mathbf{F} by expanding the determinant of $(\mathbf{A} - \lambda\mathbf{I})$ and solving the characteristic equation by Newton's method. Solve for the corresponding eigenvectors by substituting the eigenvalues into the equation $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0$ and solving for \mathbf{x} . Let the first component of \mathbf{x} be unity.

2.3 The Power Method

The Direct Power Method

3. Solve for the largest (in magnitude) eigenvalue of matrix **A** and the corresponding eigenvector \mathbf{x} by the power method. (a) Let the first component of \mathbf{x} be the unity component. (b) Let the second component of \mathbf{x} be the unity component. (c) Show that the eigenvectors obtained in parts (a) and (b) are equivalent.
4. Solve for the largest (in magnitude) eigenvalue of matrix **A** and the corresponding eigenvector \mathbf{x} by the power method with $\mathbf{x}^{(0)T} = [1.0 \ 0.0]$ and $[0.0 \ 1.0]$. (a) For each $\mathbf{x}^{(0)}$, let the first component of \mathbf{x} be the unity component. (b) For each $\mathbf{x}^{(0)}$, let the second component of \mathbf{x} be the unity component.
5. Solve Problem 3 for matrix **B**.
6. Solve Problem 4 for matrix **B**.
7. Solve Problem 3 for matrix **C**.
8. Solve Problem 4 for matrix **C**.
9. Solve for the largest (in magnitude) eigenvalue of matrix **D** and the corresponding eigenvector \mathbf{x} by the power method. (a) Let the first component of \mathbf{x} be the unity component. (b) Let the second component of \mathbf{x} be the unity component. (c) Let the third component of \mathbf{x} be the unity component. (d) Show that the eigenvectors obtained in parts (a), (b), and (c) are equivalent.
10. Solve for the largest (in magnitude) eigenvalue of matrix **D** and the corresponding eigenvector \mathbf{x} by the power method with $\mathbf{x}^{(0)T} = [1.0 \ 0.0 \ 0.0]$, $[0.0 \ 1.0 \ 0.0]$, and $[0.0 \ 0.0 \ 1.0]$. (a) For each $\mathbf{x}^{(0)}$, let the first component of \mathbf{x} be the unity component. (b) For each $\mathbf{x}^{(0)}$, let the second component of \mathbf{x} be the unity component. (c) For each $\mathbf{x}^{(0)}$, let the third component of \mathbf{x} be the unity component.
11. Solve Problem 9 for matrix **E**.
12. Solve Problem 10 for matrix **E**.
13. Solve Problem 9 for matrix **F**.
14. Solve Problem 10 for matrix **F**.
15. Solve for the largest (in magnitude) eigenvalue of matrix **G** and the corresponding eigenvector \mathbf{x} by the power method. (a) Let the first component of \mathbf{x} be the unity component. (b) Let the second component of \mathbf{x} be the unity component. (c) Let the third component of \mathbf{x} be the unity component. (d) Let the fourth component of \mathbf{x} be the unity component. (e) Show that the eigenvectors obtained in parts (a) to (d) are equivalent.
16. Solve for the largest (in magnitude) eigenvalue of matrix **G** and the corresponding eigenvector \mathbf{x} by the power method with $\mathbf{x}^{(0)T} = [1.0 \ 0.0 \ 0.0 \ 0.0]$, $[0.0 \ 1.0 \ 0.0 \ 0.0]$, $[0.0 \ 0.0 \ 1.0 \ 0.0]$, and $[0.0 \ 0.0 \ 0.0 \ 1.0]$. (a) For each $\mathbf{x}^{(0)}$, let the first component of \mathbf{x} be the unity component. (b) For each $\mathbf{x}^{(0)}$, let the second component of \mathbf{x} be the unity component. (c) For each $\mathbf{x}^{(0)}$, let the third component of \mathbf{x} be the unity component. (d) For each $\mathbf{x}^{(0)}$, let the fourth component of \mathbf{x} be the unity component.
17. Solve Problem 15 for matrix **H**.
18. Solve Problem 16 for matrix **H**.

The Inverse Power Method

19. Solve for the smallest (in magnitude) eigenvalue of matrix **A** and the corresponding eigenvector \mathbf{x} by the inverse power method using the matrix inverse. Use Gauss-Jordan elimination to find the matrix inverse. (a) Let the first component of \mathbf{x} be the unity component. (b) Let the second component of \mathbf{x} be the unity component. (c) Show that the eigenvectors obtained in parts (a) and (b) are equivalent.
20. Solve for the smallest (in magnitude) eigenvalue of matrix **A** and the corresponding eigenvector \mathbf{x} by the inverse power method using the matrix inverse with $\mathbf{x}^{(0)T} = [1.0 \ 0.0]$ and $[0.0 \ 1.0]$. (a) For each $\mathbf{x}^{(0)}$, let the first component of \mathbf{x} be the unity component. (b) For each $\mathbf{x}^{(0)}$, let the second component of \mathbf{x} be the unity component.
21. Solve Problem 19 for matrix **B**.
22. Solve Problem 20 for matrix **B**.
23. Solve Problem 19 for matrix **C**.
24. Solve Problem 20 for matrix **C**.
25. Solve for the smallest (in magnitude) eigenvalue of matrix **D** and the corresponding eigenvector \mathbf{x} by the inverse power method using the matrix inverse. Use Gauss-Jordan elimination to find the matrix inverse. (a) Let the first component of \mathbf{x} be the unity component. (b) Let the second component of \mathbf{x} be the unity component. (c) Let the third component of \mathbf{x} be the unity component. (d) Show that the eigenvectors obtained in parts (a), (b), and (c) are equivalent.
26. Solve for the smallest (in magnitude) eigenvalue of matrix **D** and the corresponding eigenvector \mathbf{x} by the inverse power method using the matrix inverse with $\mathbf{x}^{(0)T} = [1.0 \ 0.0 \ 0.0]$, $[0.0 \ 1.0 \ 0.0]$, and $[0.0 \ 0.0 \ 1.0]$. (a) For each $\mathbf{x}^{(0)}$, let the first component of \mathbf{x} be the unity component. (b) For each $\mathbf{x}^{(0)}$, let the second component of \mathbf{x} be the unity component. (c) For each $\mathbf{x}^{(0)}$, let the third component of \mathbf{x} be the unity component.
27. Solve Problem 25 for matrix **E**.
28. Solve Problem 26 for matrix **E**.
29. Solve Problem 25 for matrix **F**.
30. Solve Problem 26 for matrix **F**.
31. Solve for the smallest (in magnitude) eigenvalue of matrix **G** and the corresponding eigenvector \mathbf{x} by the inverse power method using the matrix inverse. Use Gauss-Jordan elimination to find the matrix inverse. (a) Let the first component of \mathbf{x} be the unity component. (b) Let the second component of \mathbf{x} be the unity component. (c) Let the third component of \mathbf{x} be the unity component. (d) Let the fourth component of \mathbf{x} be the unity component. (e) Show that the eigenvectors obtained in parts (a) to (d) are equivalent.
32. Solve for the smallest (in magnitude) eigenvalue of matrix **G** and the corresponding eigenvector \mathbf{x} by the inverse power method using the matrix inverse with $\mathbf{x}^{(0)T} = [1.0 \ 0.0 \ 0.0 \ 0.0]$, $[0.0 \ 1.0 \ 0.0 \ 0.0]$, $[0.0 \ 0.0 \ 1.0 \ 0.0]$, and $[0.0 \ 0.0 \ 0.0 \ 1.0]$. (a) For each $\mathbf{x}^{(0)}$, let the first component of \mathbf{x} be the unity component. (b) For each $\mathbf{x}^{(0)}$, let the second component of \mathbf{x} be the unity component. (c) For each $\mathbf{x}^{(0)}$, let the third component of \mathbf{x} be the unity component. (d) For each $\mathbf{x}^{(0)}$, let the fourth component of \mathbf{x} be the unity component.

33. Solve Problem 31 for matrix **H**.
34. Solve Problem 32 for matrix **H**.
35. Solve Problem 19 using Doolittle LU factorization.
36. Solve Problem 21 using Doolittle LU factorization.
37. Solve Problem 23 using Doolittle LU factorization.
38. Solve Problem 25 using Doolittle LU factorization.
39. Solve Problem 27 using Doolittle LU factorization.
40. Solve Problem 29 using Doolittle LU factorization.
41. Solve Problem 31 using Doolittle LU factorization.
42. Solve Problem 33 using Doolittle LU factorization.

Shifting Eigenvalues to Find the Opposite Extreme Eigenvalue

43. Solve for the smallest eigenvalue of matrix **A** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 5.0$ and applying the shifted power method. Let the first component of **x** be the unity component.
44. Solve for the smallest eigenvalue of matrix **B** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 6.0$ and applying the shifted power method. Let the first component of **x** be the unity component.
45. Solve for the smallest eigenvalue of matrix **C** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 5.0$ and applying the shifted power method. Let the first component of **x** be the unity component.
46. Solve for the smallest eigenvalue of matrix **D** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 4.5$ and applying the shifted power method. Let the first component of **x** be the unity component.
47. Solve for the smallest eigenvalue of matrix **E** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 4.0$ and applying the shifted power method. Let the first component of **x** be the unity component.
48. Solve for the smallest eigenvalue of matrix **F** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 4.0$ and applying the shifted power method. Let the first component of **x** be the unity component.
49. Solve for the smallest eigenvalue of matrix **G** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 6.6$ and applying the shifted power method. Let the first component of **x** be the unity component.
50. Solve for the smallest eigenvalue of matrix **H** and the corresponding eigenvector **x** by shifting the eigenvalues by $s = 6.8$ and applying the shifted power method. Let the first component of **x** be the unity component.

Shifting Eigenvalues to Find Intermediate Eigenvalues

51. The third eigenvalue of matrix **D** and the corresponding eigenvector **x** can be found in a trial and error manner by assuming a value for λ between the smallest (in absolute value) and largest (in absolute value) eigenvalues, shifting the matrix by that value, and applying the inverse power method to the shifted matrix. Solve for the third eigenvalue of matrix **D** by shifting by $s = 0.8$ and applying the shifted inverse power method using Doolittle LU factorization. Let the first component of **x** be the unity component.
52. Repeat Problem 51 for matrix **E** by shifting by $s = -0.4$.
53. Repeat Problem 51 for matrix **F** by shifting by $s = 0.6$.
54. The third and fourth eigenvalues of matrix **G** and the corresponding eigen-

vectors \mathbf{x} can be found in a trial and error manner by assuming a value for λ between the smallest (in absolute value) and largest (in absolute value) eigenvalues, shifting the matrix by that value, and applying the shifted inverse power method to the shifted matrix. This procedure can be quite time consuming for large matrices. Solve for these two eigenvalues by shifting \mathbf{G} by $s = 1.5$ and -0.5 and applying the shifted inverse power method using Doolittle LU factorization. Let the first component of \mathbf{x} be the unity component.

55. Repeat Problem 54 for matrix \mathbf{H} by shifting by $s = 1.7$ and -0.5 .

Shifting Eigenvalues to Accelerate Convergence

The convergence rate of an eigenproblem can be accelerated by stopping the iterative procedure after a few iterations, shifting the approximate result back to determine an improved approximation of λ , shifting the original matrix by this improved approximation of λ , and continuing with the inverse power method.

56. Apply the above procedure to Problem 46. After 10 iterations in Problem 46, $\lambda_s^{(10)} = -4.722050$ and $\mathbf{x}^{(10)T} = [1.0 \quad -1.330367 \quad 0.047476]$.
57. Apply the above procedure to Problem 47. After 20 iterations in Problem 47, $\lambda_s^{(20)} = -4.683851$ and $\mathbf{x}^{(20)T} = [0.256981 \quad 1.0 \quad -0.732794]$.
58. Apply the above procedure to Problem 48. After 10 iterations in Problem 48, $\lambda_s^{(10)} = -4.397633$ and $\mathbf{x}^{(10)T} = [1.0 \quad 9.439458 \quad -5.961342]$.
59. Apply the above procedure to Problem 49. After 20 iterations in Problem 49, $\lambda_s^{(20)} = -7.388013$ and $\mathbf{x}^{(20)T} = [1.0 \quad -0.250521 \quad -1.385861 \quad -0.074527]$.
60. Apply the above procedure to Problem 50. After 20 iterations in Problem 50, $\lambda_s^{(20)} = -8.304477$ and $\mathbf{x}^{(20)T} = [1.0 \quad -1.249896 \quad 0.587978 \quad -0.270088]$.

2.4 The Direct Method

61. Solve for the largest eigenvalue of matrix \mathbf{D} by the direct method using the secant method. Let $\lambda^{(0)} = 5.0$ and $\lambda^{(1)} = 4.0$.
62. Solve for the largest eigenvalue of matrix \mathbf{E} by the direct method using the secant method. Let $\lambda^{(0)} = 5.0$ and $\lambda^{(1)} = 4.0$.
63. Solve for the largest eigenvalue of matrix \mathbf{F} by the direct method using the secant method. Let $\lambda^{(0)} = 5.0$ and $\lambda^{(1)} = 4.0$.
64. Solve for the largest eigenvalue of matrix \mathbf{G} by the direct method using the secant method. Let $\lambda^{(0)} = 7.0$ and $\lambda^{(1)} = 6.0$.
65. Solve for the largest eigenvalue of matrix \mathbf{H} by the direct method using the secant method. Let $\lambda^{(0)} = 7.0$ and $\lambda^{(1)} = 6.0$.
66. Solve for the smallest eigenvalue of matrix \mathbf{D} by the direct method using the secant method. Let $\lambda^{(0)} = 0.0$ and $\lambda^{(1)} = -0.5$.
67. Solve for the smallest eigenvalue of matrix \mathbf{E} by the direct method using the secant method. Let $\lambda^{(0)} = -0.5$ and $\lambda^{(1)} = -1.0$.
68. Solve for the smallest eigenvalue of matrix \mathbf{F} by the direct method using the secant method. Let $\lambda^{(0)} = -0.5$ and $\lambda^{(1)} = -1.0$.
69. Solve for the smallest eigenvalue of matrix \mathbf{G} by the direct method using the secant method. Let $\lambda^{(0)} = -0.8$ and $\lambda^{(1)} = -1.0$.
70. Solve for the smallest eigenvalue of matrix \mathbf{H} by the direct method using the secant method. Let $\lambda^{(0)} = -1.1$ and $\lambda^{(1)} = -1.5$.

2.5 The QR Method

71. Solve for the eigenvalues of matrix **A** by the QR method.
72. Solve for the eigenvalues of matrix **B** by the QR method.
73. Solve for the eigenvalues of matrix **C** by the QR method.
74. Solve for the eigenvalues of matrix **D** by the QR method.
75. Solve for the eigenvalues of matrix **E** by the QR method.
76. Solve for the eigenvalues of matrix **F** by the QR method.
77. Solve for the eigenvalues of matrix **G** by the QR method.
78. Solve for the eigenvalues of matrix **H** by the QR method.

2.6 Eigenvectors

79. Solve for the eigenvectors of matrix **A** corresponding to the eigenvalues found in Problem 71 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.
80. Solve for the eigenvectors of matrix **B** corresponding to the eigenvalues found in Problem 72 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.
81. Solve for the eigenvectors of matrix **C** corresponding to the eigenvalues found in Problem 73 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.
82. Solve for the eigenvectors of matrix **D** corresponding to the eigenvalues found in Problem 74 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.
83. Solve for the eigenvectors of matrix **E** corresponding to the eigenvalues found in Problem 75 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.
84. Solve for the eigenvectors of matrix **F** corresponding to the eigenvalues found in Problem 76 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.
85. Solve for the eigenvectors of matrix **G** corresponding to the eigenvalues found in Problem 77 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.
86. Solve for the eigenvectors of matrix **H** corresponding to the eigenvalues found in Problem 78 by applying the shifted inverse power method one time. Let the first component of \mathbf{x} be the unity component.

2.8 Programs

87. Implement the direct power method program presented in Section 2.8.1. Check out the program using the given data set.
88. Solve any of Problems 3 to 18 using the direct power method program.
89. Check out the shifted direct power method of finding the opposite extreme eigenvalue using the data set specified by the *comment statements*.
90. Solve any of Problems 43 to 50 using the shifted direct power method program.
91. Implement the inverse power method program presented in Section 2.8.2. Check out the program using the given data set.

92. Solve any of Problems 19 to 34 using the inverse power method program.
93. Check out the shifted inverse power method for finding intermediate eigenvalues using the data set specified by the *comment statements*.
94. Solve any of Problems 51 to 55 using the shifted inverse power method program.
95. Check out the shifted inverse power method for accelerating convergence using the data set specified by the *comment statements*.
96. Solve any of Problems 56 to 60 using the shifted inverse power method program.
97. Check out the shifted inverse power method program for evaluating eigenvectors for a specified eigenvalue using the data set specified by the *comment statements*.
98. Solve any of Problems 79 to 86 using the shifted inverse power method program.

3

Nonlinear Equations

- 3.1. Introduction
- 3.2. General Features of Root Finding
- 3.3. Closed Domain (Bracketing) Methods
- 3.4. Open Domain Methods
- 3.5. Polynomials
- 3.6. Pitfalls of Root Finding Methods and Other Methods of Root Finding
- 3.7. Systems of Nonlinear Equations
- 3.8. Programs
- 3.9. Summary
Problems

Examples

- 3.1. Interval halving (bisection)
- 3.2. False position (regula falsi)
- 3.3. Fixed-point iteration
- 3.4. Newton's method
- 3.5. The secant method
- 3.6. Muller's method
- 3.7. Newton's method for simple roots
- 3.8. Polynomial deflation
- 3.9. Newton's method for multiple roots
- 3.10. Newton's method for complex roots
- 3.11. Bairstow's method for quadratic factors
- 3.12. Newton's method for two coupled nonlinear equations

3.1 INTRODUCTION

Consider the four-bar linkage illustrated in Figure 3.1. The angle $\alpha = \theta_4 - \pi$ is the input to this mechanism, and the angle $\phi = \theta_2$ is the output. A relationship between α and ϕ can be obtained by writing the vector loop equation:

$$\vec{r}_2 + \vec{r}_3 + \vec{r}_4 - \vec{r}_1 = 0 \quad (3.1)$$

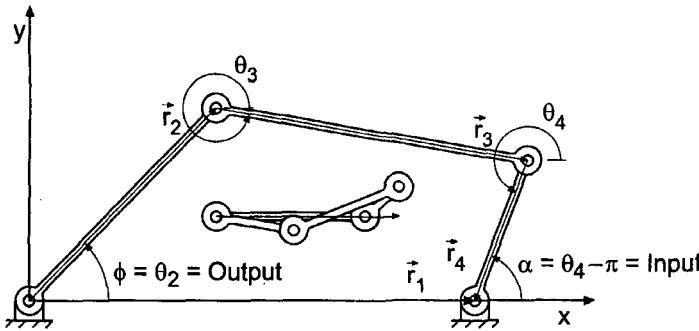


Figure 3.1 Four-bar linkage.

Let \vec{r}_1 lie along the x axis. Equation (3.1) can be written as two scalar equations, corresponding to the x and y components of the \vec{r} vectors. Thus,

$$r_2 \cos(\theta_2) + r_3 \cos(\theta_3) + r_4 \cos(\theta_4) - r_1 = 0 \quad (3.2a)$$

$$r_2 \sin(\theta_2) + r_3 \sin(\theta_3) + r_4 \sin(\theta_4) = 0 \quad (3.2b)$$

Combining Eqs. (3.2a) and (3.2b), letting $\theta_2 = \phi$ and $\theta_4 = \alpha + \pi$, and simplifying yields Freudenstein's (1955) equation:

$$R_1 \cos(\alpha) - R_2 \cos(\phi) + R_3 - \cos(\alpha - \phi) = 0 \quad (3.3)$$

where

$$R_1 = \frac{r_1}{r_2} \quad R_2 = \frac{r_1}{r_4} \quad R_3 = \frac{r_1^2 + r_2^2 + r_3^2 + r_4^2}{2r_2r_4} \quad (3.4)$$

Consider the particular four-bar linkage specified by $r_1 = 10$, $r_2 = 6$, $r_3 = 8$, and $r_4 = 4$, which is illustrated in Figure 3.1. Thus, $R_1 = \frac{5}{3}$, $R_2 = \frac{5}{2}$, $R_3 = \frac{11}{6}$, and Eq. (3.3) becomes

$$\frac{5}{3} \cos(\alpha) - \frac{5}{2} \cos(\phi) + \frac{11}{6} - \cos(\alpha - \phi) = 0 \quad (3.5)$$

The exact solution of Eq. (3.5) is tabulated in Table 3.1 and illustrated in Figure 3.2. Table 3.1 and Figure 3.2 correspond to the case where links 2, 3, and 4 are in the upper half-plane. This problem will be used throughout Chapter 3 to illustrate methods of solving for the roots of nonlinear equations. A mirror image solution is obtained for the case where links 2, 3, and 4 are in the lower half-plane. Another solution and its mirror image about the x axis are obtained if link 4 is in the upper half plane, link 2 is in the lower half-plane, and link 3 crosses the x axis, as illustrated by the small insert in Figure 3.1.

Table 3.1. Exact Solution of the Four-Bar Linkage Problem

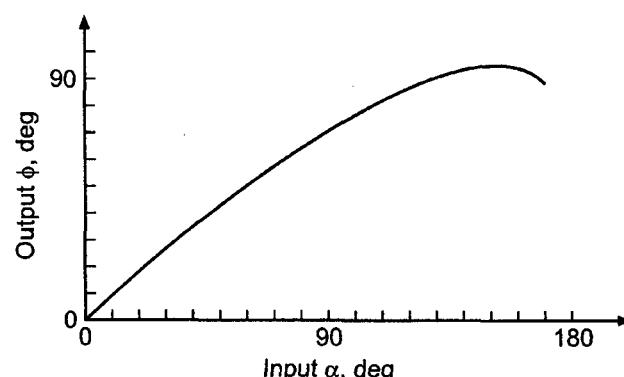
α , deg	ϕ , deg	α , deg	ϕ , deg	α , deg	ϕ , deg
0.0	0.000000	70.0	54.887763	130.0	90.124080
10.0	8.069345	80.0	62.059980	140.0	92.823533
20.0	16.113229	90.0	68.888734	150.0	93.822497
30.0	24.104946	100.0	75.270873	160.0	92.734963
40.0	32.015180	110.0	81.069445	170.0	89.306031
50.0	39.810401	120.0	86.101495	180.0	83.620630
60.0	47.450827				

Many problems in engineering and science require the solution of a nonlinear equation. The problem can be stated as follows:

Given the continuous nonlinear function $f(x)$,
find the value $x = \alpha$ such that $f(\alpha) = 0$.

Figure 3.3 illustrates the problem graphically. The nonlinear equation, $f(x) = 0$, may be an algebraic equation (i.e., an equation involving $+$, $-$, \times , $/$, and radicals), a transcendental equation (i.e., an equation involving trigonometric, logarithmic, exponential, etc., functions), the solution of a differential equation, or any nonlinear relationship between an input x and an output $f(x)$.

There are two phases to finding the roots of a nonlinear equation: *bounding the root* and *refining the root* to the desired accuracy. Two general types of root-finding methods exist: *closed domain (bracketing) methods* which bracket the root in an ever-shrinking closed interval, and *open domain (nonbracketing) methods*. Several classical methods of both types are presented in this chapter. Polynomial root finding is considered as a special case. There are numerous pitfalls in finding the roots of nonlinear equations, which are discussed in some detail.

**Figure 3.2** Exact solution of the four-bar linkage problem.

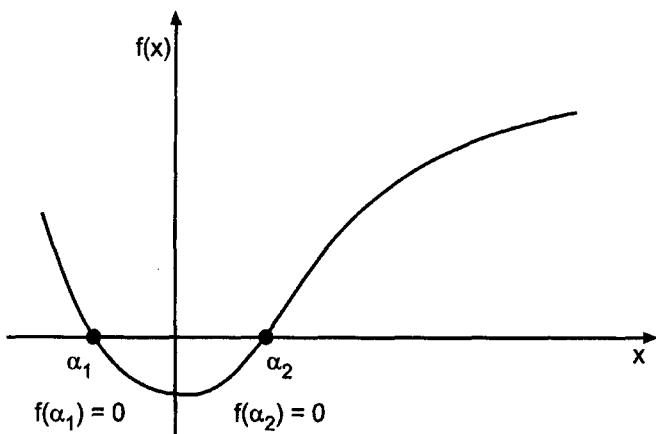


Figure 3.3 Solution of a nonlinear equation.

Figure 3.4 illustrates the organization of Chapter 3. After the introductory material presented in this section, some of the general features of root finding are discussed. The material then splits into a discussion of closed domain (bracketing) methods and open domain methods. Several special procedures applicable to polynomials are presented. After the presentation of the root finding methods, a section discussing some of the pitfalls of root finding and some other methods of root finding follows. A brief introduction to finding the roots of systems of nonlinear equations is presented. A section presenting several programs for solving nonlinear equations follows. The chapter closes with a Summary, which presents some philosophy to help you choose a specific method for a particular problem and lists the things you should be able to do after studying Chapter 3.

3.2 GENERAL FEATURES OF ROOT FINDING

Solving for the zeros of an equation, a process known as *root finding*, is one of the oldest problems in mathematics. Some general features of root finding are discussed in this section.

There are two distinct phases in finding the roots of a nonlinear equation: (1) *bounding the solution* and (2) *refining the solution*. These two phases are discussed in Sections 3.2.1 and 3.2.2, respectively. In general, nonlinear equations can behave in many different ways in the vicinity of a root. Typical behaviors are discussed in Section 3.2.3. Some general philosophy of root finding is discussed in Section 3.2.4.

3.2.1. Bounding the Solution

Bounding the solution involves finding a rough estimate of the solution that can be used as the initial approximation, or the starting point, in a systematic procedure that refines the solution to a specified tolerance in an efficient manner. If possible, the root should be bracketed between two points at which the value of the nonlinear function has opposite signs. Several possible bounding procedures are:

1. Graphing the function
2. Incremental search

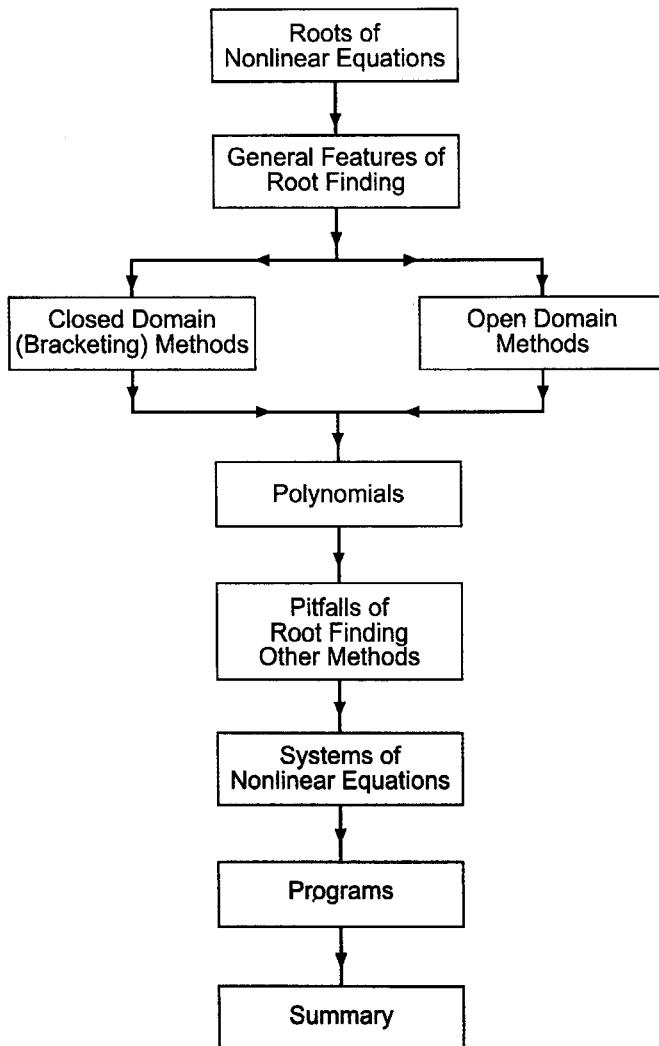


Figure 3.4 Organization of Chapter 3.

3. Past experience with the problem or a similar problem
4. Solution of a simplified approximate model
5. Previous solution in a sequence of solutions

Graphing the function involves plotting the nonlinear function over the range of interest. Many hand calculators have the capability to graph a function simply by defining the function and specifying the range of interest. Spreadsheets generally have graphing capability, as does software like Matlab and Mathcad. Very little effort is required. The resolution of the plots is generally not precise enough for an accurate result. However, the results are generally accurate enough to bound the solution. Plots of a nonlinear function display the general behavior of the nonlinear equation and permit the anticipation of problems.

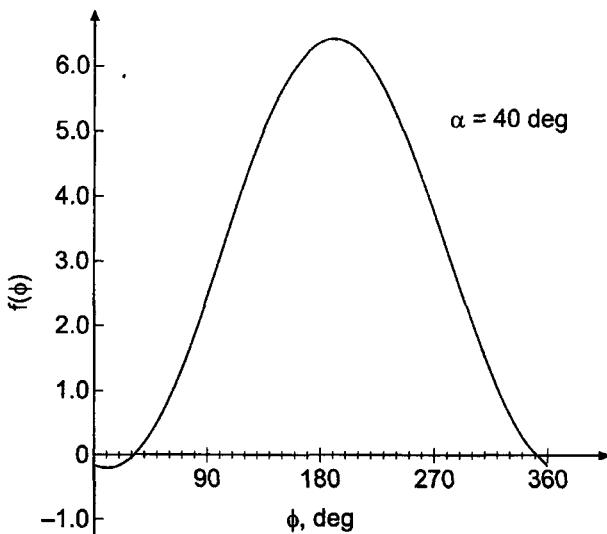


Figure 3.5 Graph of Eq. (3.5) for $\alpha = 40 \text{ deg}$.

As an example of graphing a function to bound a root, consider the four-bar linkage problem presented in Section 3.1. Consider an input of $\alpha = 40 \text{ deg}$. The graph of Eq. (3.5) with $\alpha = 40 \text{ deg}$ is presented in Figure 3.5. The graph shows that there are two roots of Eq. (3.5) when $\alpha = 40 \text{ deg}$: one root between $\phi = 30 \text{ deg}$ and $\phi = 40 \text{ deg}$, and one root between $\phi = 350$ (or -10 deg) and $\phi = 360$ (or 0 deg).

An *incremental search* is conducted by starting at one end of the region of interest and evaluating the nonlinear function at small increments across the region. When the value of the function changes sign, it is assumed that a root lies in that interval. The two end points of the interval containing the root can be used as initial guesses for a refining method. If multiple roots are suspected, check for sign changes in the derivative of the function between the ends of the interval.

To illustrate an incremental search, let's evaluate Eq. (3.5) with $\alpha = 40 \text{ deg}$ for ϕ from 0 to 360 deg for $\Delta\phi = 10 \text{ deg}$. The results are presented in Table 3.2. The same two roots identified by graphing the function are located.

Table 3.2. Incremental Search for Eq. (3.5) with $\alpha = 40 \text{ deg}$

ϕ , deg	$f(\phi)$						
0.0	-0.155970	100.0	3.044194	190.0	6.438119	280.0	3.175954
10.0	-0.217971	110.0	3.623104	200.0	6.398988	290.0	2.597044
20.0	-0.178850	120.0	4.186426	210.0	6.259945	300.0	2.033722
30.0	-0.039797	130.0	4.717043	220.0	6.025185	310.0	1.503105
40.0	0.194963	140.0	5.198833	230.0	5.701851	320.0	1.021315
50.0	0.518297	150.0	5.617158	240.0	5.299767	330.0	0.602990
60.0	0.920381	160.0	5.959306	250.0	4.831150	340.0	0.260843
70.0	1.388998	170.0	6.214881	260.0	4.310239	350.0	0.005267
80.0	1.909909	180.0	6.376119	270.0	3.752862	360.0	-0.155970
90.0	2.467286						

Whatever procedure is used to bound the solution, the initial approximation must be sufficiently close to the exact solution to ensure (a) that the systematic refinement procedure converges, and (b) that the solution converges to the desired root of the nonlinear equation.

3.2.2. Refining the Solution

Refining the solution involves determining the solution to a specified tolerance by an efficient systematic procedure. Several methods for refining the solution are:

1. Trial and error
2. Closed domain (bracketing) methods
3. Open domain methods

Trial and error methods simply guess the root, $x = \alpha$, evaluate $f(\alpha)$, and compare to zero. If $f(\alpha)$ is close enough to zero, quit. If not, guess another α , and continue until $f(\alpha)$ is close enough to zero. This approach is totally unacceptable.

Closed domain (bracketing) methods are methods that start with two values of x which bracket the root, $x = \alpha$, and systematically reduce the interval while keeping the root trapped within the interval. Two such methods are presented in Section 3.3:

1. Interval halving (bisection)
2. False position (regula falsi)

Bracketing methods are robust in that they are guaranteed to obtain a solution since the root is trapped in the closed interval. They can be slow to converge.

Open domain methods do not restrict the root to remain trapped in a closed interval. Consequently, they are not as robust as bracketing methods and can actually diverge. However, they use information about the nonlinear function itself to refine the estimates of the root. Thus, they are considerably more efficient than bracketing methods. Four open domain methods are presented in Section 3.4:

1. The fixed-point iteration method
2. Newton's method
3. The secant method
4. Muller's method

3.2.3. Behavior of Nonlinear Equations

Nonlinear equations can behave in various ways in the vicinity of a root. Algebraic and transcendental equations may have distinct (i.e., simple) real roots, repeated (i.e., multiple) real roots, or complex roots. Polynomials may have real or complex roots. If the polynomial coefficients are all real, complex roots occur in conjugate pairs. If the polynomial coefficients are complex, single complex roots can occur.

Figure 3.6 illustrates several distinct types of behavior of nonlinear equations in the vicinity of a root. Figure 3.6a illustrates the case of a single real root, which is called a *simple root*. Figure 3.6b illustrates a case where no real roots exist. Complex roots may

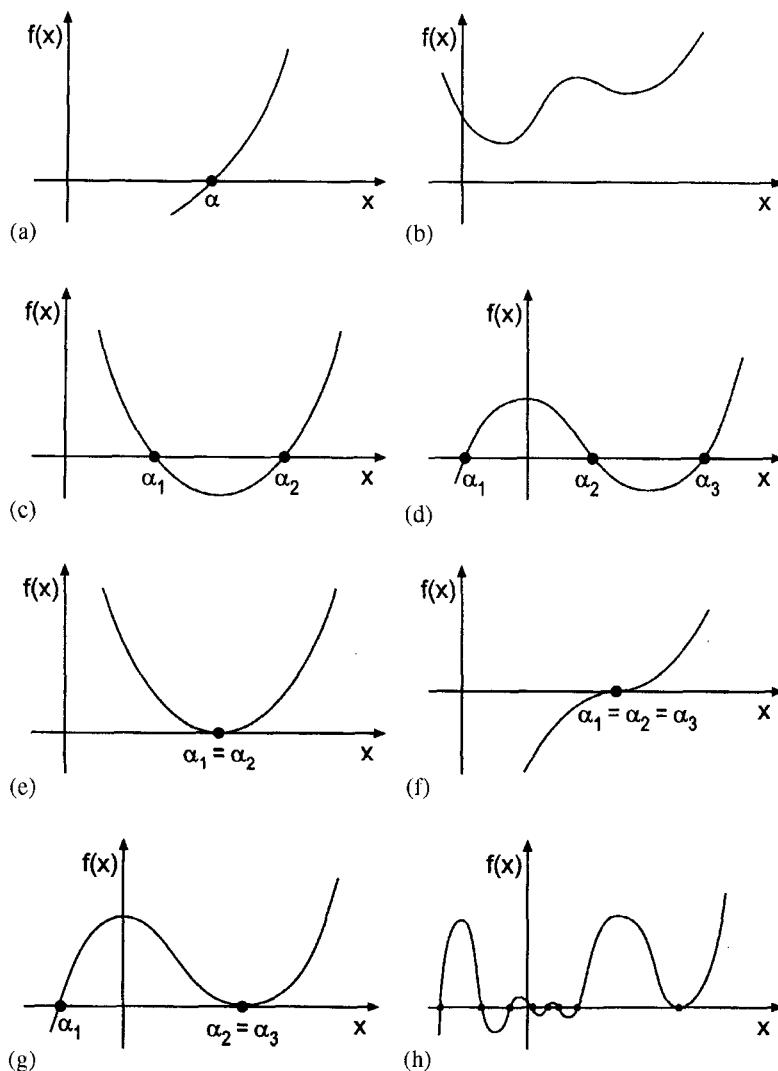


Figure 3.6 Solution behavior. (a) Simple root. (b) No real roots. (c) Two simple roots. (d) Three simple roots. (e) Two multiple roots. (f) Three multiple roots. (g) One simple and two multiple roots. (h) General case.

exist in such a case. Situations with two and three simple roots are illustrated in Figure 3.6c and d, respectively. Situations with two and three multiple roots are illustrated in Figure 3.6e and f, respectively. A situation with one simple root and two multiple roots is illustrated in Figure 3.6g. Lastly, Figure 3.6h illustrates the general case where any number of simple or multiple roots can exist.

Many problems in engineering and science involve a simple root, as illustrated in Figure 3.6a. Almost any root-finding method can find such a root if a reasonable initial approximation is furnished. In the other situations illustrated in Figure 3.6, extreme care may be required to find the desired roots.

3.2.4. Some General Philosophy of Root Finding

There are numerous methods for finding the roots of a nonlinear equation. The roots have specific values, and the method used to find the roots does not affect the values of the roots. However, the method can determine whether or not the roots can be found and the amount of work required to find them. Some general philosophy of root finding is presented below.

1. Bounding methods should bracket a root, if possible.
2. Good initial approximations are extremely important.
3. Closed domain methods are more robust than open domain methods because they keep the root bracketed in a closed interval.
4. Open domain methods, when they converge, generally converge faster than closed domain methods.
5. For smoothly varying functions, most algorithms will always converge if the initial approximation is close enough. The rate of convergence of most algorithms can be determined in advance.
6. Many, if not most, problems in engineering and science are well behaved and straightforward. In such cases, a straightforward open domain method, such as Newton's method presented in Section 3.4.2 or the secant method presented in Section 3.4.3, can be applied without worrying about special cases and peculiar behavior. If problems arise during the solution, then the peculiarities of the nonlinear equation and the choice of solution method can be reevaluated.
7. When a problem is to be solved only once or a few times, the efficiency of the method is not of major concern. However, when a problem is to be solved many times, efficiency of the method is of major concern.
8. Polynomials can be solved by any of the methods for solving nonlinear equations. However, the special techniques applicable to polynomials should be considered.
9. If a nonlinear equation has complex roots, that must be anticipated when choosing a method.
10. Analyst's time versus computer time must be considered when selecting a method.
11. Blanket generalizations about root-finding methods are generally not possible.

Root-finding algorithms should contain the following features:

1. An upper limit on the number of iterations.
2. If the method uses the derivative $f'(x)$, it should be monitored to ensure that it does not approach zero.
3. A convergence test for the change in the magnitude of the solution, $|x_{i+1} - x_i|$, or the magnitude of the nonlinear function, $|f(x_{i+1})|$, must be included.
4. When convergence is indicated, the final root estimate should be inserted into the nonlinear function $f(x)$ to guarantee that $f(x) = 0$ within the desired tolerance.

3.3 CLOSED DOMAIN (BRACKETING) METHODS

Two of the simplest methods for finding the roots of a nonlinear equation are:

1. Interval halving (bisection)
2. False position (regula falsi)

In these two methods, two estimates of the root which bracket the root must first be found by the bounding process. The root, $x = \alpha$, is bracketed by the two estimates. The objective is to locate the root to within a specified tolerance by a systematic procedure while keeping the root bracketed. Methods which keep the root bracketed during the refinement process are called *closed domain*, or *bracketing methods*.

3.3.1. Interval Halving (Bisection)

One of the simplest methods for finding a root of a nonlinear equation is *interval halving* (also known as *bisection*). In this method, two estimates of the root, $x = a$ to the left of the root and $x = b$ to the right of the root, which bracket the root, must first be obtained, as illustrated in Figure 3.7, which illustrates the two possibilities with $f'(x) > 0$ and $f'(x) < 0$. The root, $x = \alpha$, obviously lies between a and b , that is, in the interval (a, b) . The interval between a and b can be halved by averaging a and b . Thus, $c = (a + b)/2$. There are now two intervals: (a, c) and (c, b) . The interval containing the root, $x = \alpha$, depends on the value of $f(c)$. If $f(a)f(c) < 0$, which is the case in Figure 3.7a, the root is in the interval (a, c) . Thus, set $b = c$ and continue. If $f(a)f(c) > 0$, which is the case in Figure 3.7b, the root is in the interval (c, b) . Thus, set $a = c$ and continue. If $f(a)f(c) = 0$, c is the root. Terminate the iteration. The algorithm is as follows:

$$c = \frac{a + b}{2} \quad (3.6)$$

$$\text{If } f(a)f(c) < 0: \quad a = a \text{ and } b = c \quad (3.7a)$$

$$\text{If } f(a)f(c) > 0: \quad a = c \text{ and } b = b \quad (3.7b)$$

Interval halving is an iterative procedure. The solution is not obtained directly by a single calculation. Each application of Eqs. (3.6) and (3.7) is an iteration. The iterations are continued until the size of the interval decreases below a prespecified tolerance ε_1 , that is, $|b_i - a_i| \leq \varepsilon_1$, or the value of $f(x)$ decreases below a prespecified tolerance ε_2 , that is, $|f(c_i)| \leq \varepsilon_2$, or both.

If a nonlinear equation, such as $f(x) = 1/(x - d)$ which has a singularity at $x = d$, is bracketed between a and b , interval halving will locate the discontinuity, $x = d$. A check on $|f(x)|$ as $x \rightarrow d$ would indicate that a discontinuity, not a root, is being found.

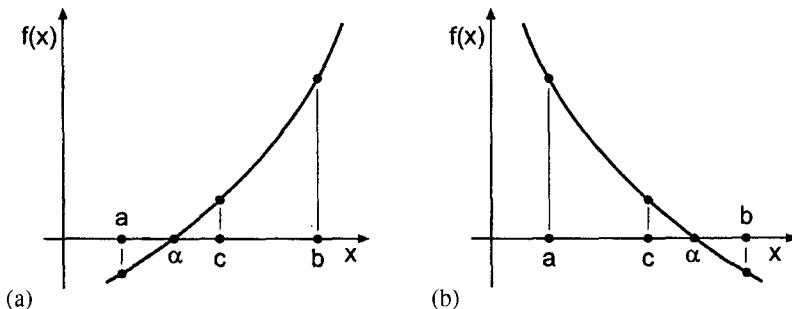


Figure 3.7 Interval halving (bisection).

Example 3.1. Interval halving (bisection).

Let's solve the four-bar linkage problem presented in Section 3.1 for an input of $\alpha = 40 \text{ deg}$ by interval halving. In calculations involving trigonometric functions, the angles must be expressed in radians. However, degrees (i.e., deg) are a more common unit of angular measure. Consequently, in all of the examples in this chapter, angles are expressed in degrees in the equations and in the tabular results, but the calculations are performed in radians. Recall Eq. (3.5) with $\alpha = 40.0 \text{ deg}$:

$$f(\phi) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(\phi) + \frac{11}{6} - \cos(40.0 - \phi) = 0.0 \quad (3.8)$$

From the bounding procedure presented in Section 3.2, let $\phi_a = 30.0 \text{ deg}$ and $\phi_b = 40.0 \text{ deg}$. From Eq. (3.8),

$$\begin{aligned} f(\phi_a) &= f(30.0) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(30.0) + \frac{11}{6} - \cos(40.0 - 30.0) \\ &= -0.03979719 \end{aligned} \quad (3.9a)$$

$$\begin{aligned} f(\phi_b) &= f(40.0) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(40.0) + \frac{11}{6} - \cos(40.0 - 40.0) \\ &= 0.19496296 \end{aligned} \quad (3.9b)$$

Thus, $\phi_a = 30.0 \text{ deg}$ and $\phi_b = 40.0 \text{ deg}$ bracket the solution. From Eq. (3.6),

$$\phi_c = \frac{\phi_a + \phi_b}{2} = \frac{30.0 + 40.0}{2} = 35.0 \text{ deg} \quad (3.10)$$

Substituting $\phi_c = 35.0 \text{ deg}$ into Eq. (3.8) yields

$$f(\phi_c) = f(35.0) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(35.0) + \frac{11}{6} - \cos(40.0 - 35.0) = 0.06599926 \quad (3.11)$$

Since $f(\phi_a)f(\phi_c) < 0$, $\phi_b = \phi_c$ for the next iteration and ϕ_a remains the same.

The solution is presented in Table 3.3. The convergence criterion is $|\phi_a - \phi_b| \leq 0.000001 \text{ deg}$, which requires 24 iterations. Clearly, convergence is rather slow. The results presented in Table 3.3 were obtained on a 13-digit precision computer.

Table 3.3. Interval Halving (Bisection)

i	ϕ_a, deg	$f(\phi_a)$	ϕ_b, deg	$f(\phi_b)$	ϕ_c, deg	$f(\phi_c)$
1	30.0	-0.03979719	40.0	0.19496296	35.0	0.06599926
2	30.0	-0.03979719	35.0	0.06599926	32.50	0.01015060
3	30.0	-0.03979719	32.50	0.01015060	31.250	-0.01556712
4	31.250	-0.01556712	32.50	0.01015060	31.8750	-0.00289347
5	31.8750	-0.00289347	32.50	0.01015060	32.18750	0.00358236
6	31.8750	-0.00289347	32.18750	0.00358236	32.031250	0.00033288
7	31.8750	-0.00289347	32.031250	0.00033288	31.953125	-0.00128318
...
22	32.015176	-0.00000009	32.015181	0.00000000	32.015178	-0.00000004
23	32.015178	-0.00000004	32.015181	0.00000000	32.015179	-0.00000002
24	32.015179	-0.00000002	32.015181	0.00000000	32.015180	-0.00000001
	32.015180	-0.00000001				

The results in the table are rounded in the sixth digit after the decimal place. The final solution agrees with the exact solution presented in Table 3.1 to six digits after the decimal place.

The interval halving (bisection) method has several advantages:

1. The root is bracketed (i.e., trapped) within the bounds of the interval, so the method is guaranteed to converge.
2. The maximum error in the root is $|b_n - a_n|$.
3. The number of iterations n , and thus the number of function evaluations, required to reduce the initial interval, $(b_0 - a_0)$, to a specified interval, $(b_n - a_n)$, is given by

$$(b_n - a_n) = \frac{1}{2^n} (b_0 - a_0) \quad (3.12)$$

since each iteration reduces the interval size by a factor of 2. Thus, n is given by

$$n = \frac{1}{\log(2)} \log\left(\frac{b_0 - a_0}{b_n - a_n}\right) \quad (3.13)$$

The major disadvantage of the interval halving (bisection) method is that the solution converges slowly. That is, it can take a large number of iterations, and thus function evaluations, to reach the convergence criterion.

3.3.2. False Position (Regula Falsi)

The interval-halving (bisection) method brackets a root in the interval (a, b) and approximates the root as the midpoint of the interval. In the *false position (regula falsi) method*, the nonlinear function $f(x)$ is assumed to be a linear function $g(x)$ in the interval (a, b) , and the root of the linear function $g(x)$, $x = c$, is taken as the next approximation of the root of the nonlinear function $f(x)$, $x = \alpha$. The process is illustrated graphically in Figure 3.8. This method is also called the *linear interpolation method*. The root of the linear function $g(x)$, that is, $x = c$, is not the root of the nonlinear function $f(x)$. It is a false position (in Latin, *regula falsi*), which gives the method its name. We now have two intervals, (a, c) and (c, b) . As in the interval-halving (bisection) method, the interval containing the root of the nonlinear function $f(x)$ is retained, as described in Section 3.3.1, so the root remains bracketed.

The equation of the linear function $g(x)$ is

$$\frac{f(c) - f(b)}{c - b} = g'(x) \quad (3.14)$$

where $f(c) = 0$, and the slope of the linear function $g'(x)$ is given by

$$g'(x) = \frac{f(b) - f(a)}{b - a} \quad (3.15)$$

Solving Eq. (3.14) for the value of c which gives $f(c) = 0$ yields

$$c = b - \frac{f(b)}{g'(x)}$$

(3.16)

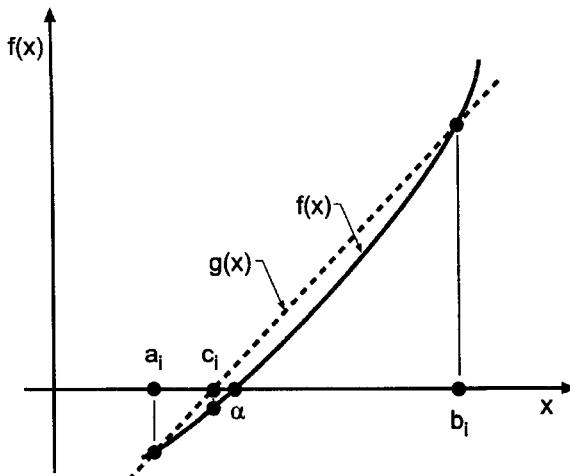


Figure 3.8 False position (regula falsi).

Note that $f(a)$ and a could have been used in Eqs. (3.14) and (3.16) instead of $f(b)$ and b . Equation (3.16) is applied repetitively until either one or both of the following two convergence criteria are satisfied:

$$|b - a| \leq \varepsilon_1 \quad \text{and/or} \quad |f(c)| \leq \varepsilon_2 \quad (3.17)$$

Example 3.2. False position (regula falsi).

As an example of the false position (regula falsi) method, let's solve the four-bar linkage problem presented in Section 3.1. Recall Eq. (3.5) with $\alpha = 40.0$ deg:

$$f(\phi) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(\phi) + \frac{11}{6} - \cos(40.0 - \phi) = 0.0 \quad (3.18)$$

Let $\phi_a = 30.0$ deg and $\phi_b = 40.0$ deg. From Eq. (3.18),

$$\begin{aligned} f(\phi_a) &= f(30.0) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(30.0) + \frac{11}{6} - \cos(40.0 - 30.0) \\ &= -0.03979719 \end{aligned} \quad (3.19a)$$

$$\begin{aligned} f(\phi_b) &= f(40.0) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(40.0) + \frac{11}{6} - \cos(40.0 - 40.0) \\ &= 0.19496296 \end{aligned} \quad (3.19b)$$

Thus, $\phi_a = 30.0$ deg and $\phi_b = 40.0$ deg bracket the solution. From Eq. (3.15),

$$g'(\phi_b) = \frac{0.19496296 - (-0.03979719)}{40.0 - 30.0} = 0.02347602 \quad (3.20)$$

Substituting these results into Eq. (3.16) yields

$$\phi_c = \phi_b - \frac{f(\phi_b)}{g'(\phi_b)} = 40.0 - \frac{0.19496296}{0.02347602} = 31.695228 \text{ deg} \quad (3.21)$$

Table 3.4. False Position (Regula Falsi)

<i>i</i>	ϕ_a , deg	$f(\phi_a)$	ϕ_b , deg	$f(\phi_b)$	ϕ_c , deg	$f(\phi_c)$
1	30.0	-0.03979719	40.0	0.19496296	31.695228	-0.00657688
2	31.695228	-0.00657688	40.0	0.19496296	31.966238	-0.00101233
3	31.966238	-0.00101233	40.0	0.19496296	32.007738	-0.00015410
4	32.007738	-0.00015410	40.0	0.19496296	32.014050	-0.00002342
5	32.014050	-0.00002342	40.0	0.19496296	32.015009	-0.00000356
6	32.015009	-0.00000356	40.0	0.19496296	32.015154	-0.00000054
7	32.015154	-0.00000054	40.0	0.19496296	32.015176	-0.00000008
8	32.015176	-0.00000008	40.0	0.19496296	32.015180	-0.00000001
9	32.015180	-0.00000001	40.0	0.19496296	32.015180	-0.00000000
	32.015180	0.00000000				

Substituting ϕ_c into Eq. (3.18) gives

$$f(\phi_c) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(31.695228) + \frac{11}{6} - \cos(40.0 - 31.695228) = -0.00657688 \quad (3.22)$$

Since $f(\phi_a)f(\phi_c) > 0.0$, ϕ_a is set equal to ϕ_c and ϕ_b remains the same. This choice of ϕ_a and ϕ_b keeps the root bracketed.

These results and the results of subsequent iterations are presented in Table 3.4. The convergence criterion, $|\phi_b - \phi_a| \leq 0.000001$ deg, is satisfied on the ninth iteration. Notice that ϕ_b does not change in this example. The root is approached monotonically from the left. This type of behavior is common for the false position method.

3.3.3. Summary

Two closed domain (bracketing) methods for finding the roots of a nonlinear equation are presented in this section: interval halving (bisection) and false position (regula falsi). Both of these methods are guaranteed to converge because they keep the root bracketed within a continually shrinking closed interval. The interval-halving method gives an exact upper bound on the error of the solution, the interval size. It converges rather slowly. The false position method generally converges more rapidly than the interval halving method, but it does not give a bound on the error of the solution.

Both methods are quite robust, but converge slowly. The open domain methods presented in Section 3.4 are generally preferred because they converge much more rapidly. However, they do not keep the root bracketed, and thus, they may diverge. In such cases, the more slowly converging bracketing methods may be preferred.

3.4 OPEN DOMAIN METHODS

The interval halving (bisection) method and the false position (regula falsi) method presented in Section 3.3 converge slowly. More efficient methods for finding the roots of a nonlinear equation are desirable. Four such methods are presented in this section:

1. Fixed-point iteration
2. Newton's method

3. The secant method
4. Muller's method

These methods are called *open domain methods* since they are not required to keep the root bracketed in a closed domain during the refinement process.

Fixed-point iteration is not a reliable method and is not recommended for use. It is included simply for completeness since it is a well-known method. Muller's method is similar to the secant method. However, it is slightly more complicated, so the secant method is generally preferred. Newton's method and the secant method are two of the most efficient methods for refining the roots of a nonlinear equation.

3.4.1. Fixed-Point Iteration

The procedure known as *fixed-point iteration* involves solving the problem $f(x) = 0$ by rearranging $f(x)$ into the form $x = g(x)$, then finding $x = \alpha$ such that $\alpha = g(\alpha)$, which is equivalent to $f(\alpha) = 0$. The value of x such that $x = g(x)$ is called a *fixed point* of the relationship $x = g(x)$. Fixed-point iteration essentially solves two functions simultaneously: $x(x)$ and $g(x)$. The point of intersection of these two functions is the solution to $x = g(x)$, and thus to $f(x) = 0$. This process is illustrated in Figure 3.9.

Since $g(x)$ is also a nonlinear function, the solution must be obtained iteratively. An initial approximation to the solution x_1 must be determined by a bounding method. This value is substituted into the function $g(x)$ to determine the next approximation. The algorithm is as follows:

$$x_{i+1} = g(x_i) \quad (3.23)$$

The procedure is repeated (iterated) until a convergence criterion is satisfied. For example,

$$|x_{i+1} - x_i| \leq \varepsilon_1 \quad \text{and/or} \quad |f(x_{i+1})| \leq \varepsilon_2 \quad (3.24)$$

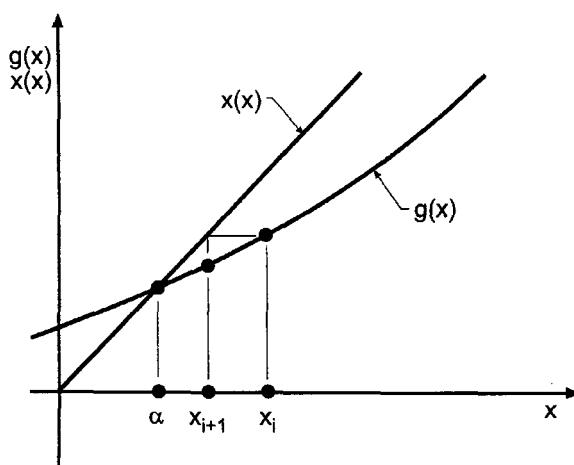


Figure 3.9 Fixed-point iteration.

Example 3.3. Fixed-point iteration.

Let's solve the four-bar linkage problem presented in Section 3.1 by fixed-point iteration. Recall Eq. (3.3):

$$f(\phi) = R_1 \cos(\alpha) - R_2 \cos(\phi) + R_3 - \cos(\alpha - \phi) = 0 \quad (3.25)$$

Equation (3.25) can be rearranged into the form $\phi = g(\phi)$ by separating the term $\cos(\alpha - \phi)$ and solving for ϕ . Thus,

$$\phi = \alpha - \cos^{-1}[R_1 \cos(\alpha) - R_2 \cos(\phi) + R_3] = \alpha - \cos^{-1}[u(\phi)] = g(\phi) \quad (3.26)$$

where

$$u(\phi) = R_1 \cos(\alpha) - R_2 \cos(\phi) + R_3 \quad (3.27)$$

The derivative of $g(\phi)$, that is, $g'(\phi)$, is of interest in the analysis of convergence presented at the end of this section. Recall

$$d(\cos^{-1}(u)) = -\frac{1}{\sqrt{1-u^2}} du \quad (3.28)$$

Differentiating Eq. (3.26) gives

$$g'(\phi) = -\frac{1}{\sqrt{1-u^2}} \frac{du}{d\phi} \quad (3.29)$$

which yields

$$g'(\phi) = \frac{R_2 \sin(\phi)}{\sqrt{1-u^2}} \quad (3.30)$$

For the four-bar linkage problem presented in Section 3.1, $R_1 = \frac{5}{3}$, $R_2 = \frac{5}{2}$, and $R_3 = \frac{11}{6}$. Let's find the output ϕ for an input $\alpha = 40$ deg. Equations (3.27), (3.26), and (3.30) become

$$u(\phi_i) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(\phi_i) + \frac{11}{6} \quad (3.31)$$

$$\phi_{i+1} = g(\phi_i) = 40.0 - \cos^{-1}[u(\phi_i)] \quad (3.32)$$

$$g'(\phi_i) = \frac{5}{2} \frac{\sin(\phi_i)}{\sqrt{1-[u(\phi_i)]^2}} \quad (3.33)$$

Let $\phi_1 = 30.0$ deg. Substituting $\phi_1 = 30.0$ deg into Eq. (3.25) gives $f(\phi_1) = -0.03979719$. Equations (3.31) to (3.33) give

$$u(30.0) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(30.0) + \frac{11}{6} = 0.945011 \quad (3.34)$$

$$\phi_2 = 40.0 - \cos^{-1}(0.945011) = 20.910798 \text{ deg} \quad (3.35)$$

$$g'(20.910798) = \frac{5}{2} \frac{\sin(20.910798)}{\sqrt{1-(0.945011)^2}} = 2.728369 \quad (3.36)$$

Substituting $\phi_2 = 20.910798$ deg into Eq. (3.25) gives $f(\phi_2) = -0.17027956$. The entire procedure is now repeated with $\phi_2 = 20.910798$ deg.

Table 3.5. Fixed-Point Iteration for the First Form of $g(\phi)$

i	ϕ_i , deg	$f(\phi_i)$	$u(\phi_i)$	$g'(\phi_i)$	ϕ_{i+1} , deg	$f(\phi_{i+1})$
1	30.000000	-0.03979719	0.94501056	2.548110	20.910798	-0.17027956
2	20.910798	-0.17027956	0.77473100	0.940796	0.780651	-0.16442489
3	0.780651	-0.16442489	0.61030612	0.028665	-12.388360	0.05797824
4	-12.388360	0.05797824	0.66828436	-0.480654	-8.065211	-0.03348286
5	-8.065211	-0.03348286	0.63480150	-0.302628	-10.594736	0.01789195
...
29	-9.747104	0.00000002	0.64616253	-0.369715	-9.747106	0.00000001
30	-9.747106	-0.00000001	0.64616254	-0.369715	-9.747106	-0.00000001
	-9.747105	-0.00000001				

These results and the results of 29 more iterations are summarized in Table 3.5. The solution has converged to the undesired root illustrated by the small insert inside the linkage illustrated in Figure 3.1. This configuration cannot be reached if the linkage is connected as illustrated in the large diagram. However, this configuration could be reached by reconnecting the linkage as illustrated in the small insert.

The results presented in Table 3.5 illustrate the major problem of open (nonbracketing) methods. Even though the desired root is bracketed by the two initial guesses, 30 deg and 40 deg, the method converged to a root outside that interval.

Let's rework the problem by rearranging Eq. (3.25) into another form of $\phi = g(\phi)$ by separating the term $R_2 \cos(\phi)$ and solving for ϕ . Thus,

$$\phi = \cos^{-1} \left\{ \frac{1}{R_2} [R_1 \cos(\alpha) + R_3 - \cos(\alpha - \phi)] \right\} = \cos^{-1}[u(\phi)] = g(\phi) \quad (3.37)$$

where

$$u(\phi) = \frac{1}{R_2} [R_1 \cos(\alpha) + R_3 - \cos(\alpha - \phi)] \quad (3.38)$$

The derivative of $g(\phi)$ is

$$g'(\phi) = \frac{\sin(\alpha - \phi)}{R_2 \sqrt{1 - u^2}} \quad (3.39)$$

For the four-bar linkage problem presented in Section 3.1, $R_1 = \frac{5}{3}$, $R_2 = \frac{5}{2}$, and $R_3 = \frac{11}{6}$. Let's find the output ϕ for the input $\alpha = 40$ deg. Equations (3.38), (3.37), and (3.39) become

$$u(\phi) = \frac{2}{5} [\frac{5}{3} \cos(40.0) + \frac{11}{6} - \cos(40.0 - \phi)] \quad (3.40)$$

$$\phi_{i+1} = g(\phi_i) = \cos^{-1}[u(\phi_i)] \quad (3.41)$$

$$g'(\phi_i) = \frac{2 \sin(40.0 - \phi_i)}{5 \sqrt{1 - [u(\phi_i)]^2}} \quad (3.42)$$

Table 3.6. Fixed-Point Iteration for the Second Form of $g(\phi)$

i	ϕ_i , deg	$f(\phi_i)$	$u(\phi_i)$	$g'(\phi)$	ϕ_{i+1} deg	$f(\phi_{i+1})$
1	30.000000	-0.03979719	0.85010653	0.131899	31.776742	-0.00491050
2	31.776742	-0.00491050	0.84814233	0.107995	31.989810	-0.00052505
3	31.989810	-0.00052505	0.84793231	0.105148	32.012517	-0.00005515
4	32.012517	-0.00005515	0.84791025	0.104845	32.014901	-0.00000578
5	32.014901	-0.00000578	0.84790794	0.104814	32.015151	-0.00000061
6	32.015151	-0.00000061	0.84790769	0.104810	32.015177	-0.00000006
7	32.015177	-0.00000006	0.84790767	0.104810	32.015180	-0.00000001
8	32.015180	-0.00000001	0.84790767	0.104810	32.015180	-0.00000000
	32.015180					

Let $\phi_1 = 30.0$ deg. Substituting $\phi_1 = 30.0$ deg into Eq. (3.25) gives $f(\phi_1) = -0.03979719$. Equations (3.40) to (3.42) give

$$u(30.0) = \frac{2}{5} \left[\frac{5}{3} \cos(40.0) + \frac{11}{6} - \cos(40.0 - 30.0) \right] = 0.850107 \quad (3.43)$$

$$\phi_2 = g(\phi_1) = g(30.0) = \cos^{-1}[u(30.0)] = \cos^{-1}(0.850107) = 31.776742 \text{ deg} \quad (3.44)$$

$$g'(30.0) = \frac{2}{5} \frac{\sin(40.0 - 30.0)}{\sqrt{1 - (0.850107)^2}} = 0.131899 \quad (3.45)$$

Substituting $\phi_2 = 31.776742$ into Eq. (3.25) gives $f(\phi_2) = -0.00491050$. The entire procedure is now repeated with $\phi_2 = 31.776742$ deg.

These results and the results of the subsequent iterations are presented in Table 3.6. The convergence criterion is $|\phi_{i+1} - \phi_i| \leq 0.000001$ deg, which requires eight iterations. This is a considerable improvement over the interval halving (bisection) method presented in Example 3.1, which requires 24 iterations. It is comparable to the false position (regula falsi) method presented in Example 3.2, which requires nine iterations.

Convergence of the fixed-point iteration method, or any iterative method of the form $x_{i+1} = g(x_i)$, is analyzed as follows. Consider the iteration formula:

$$x_{i+1} = g(x_i) \quad (3.46)$$

Let $x = \alpha$ denote the solution and let $e = (x - \alpha)$ denote the error. Subtracting $\alpha = g(\alpha)$ from Eq. (3.46) gives

$$x_{i+1} - \alpha = e_{i+1} = g(x_i) - g(\alpha) \quad (3.47)$$

Expressing $g(\alpha)$ in a Taylor series about x_i gives:

$$g(\alpha) = g(x_i) + g'(\xi)(\alpha - x_i) + \dots \quad (3.48)$$

where $x_i \leq \xi \leq \alpha$. Truncating Eq. (3.48) after the first-order term, solving for $[g(x_i) - g(\alpha)]$, and substituting the result into Eq. (3.47) yields

$$e_{i+1} = g'(\xi)e_i \quad (3.49)$$

Equation (3.49) can be used to determine whether or not a method is convergent, and if it is convergent, its rate of convergence. For any iterative method to converge,

$$\left| \frac{e_{i+1}}{e_i} \right| = |g'(\xi)| < 1 \quad (3.50)$$

Consequently, the fixed-point iteration method is convergent only if $|g'(\xi)| < 1$. Convergence is linear since e_{i+1} is linearly dependent on e_i . If $|g'(\xi)| > 1$, the procedure diverges. If $|g'(\xi)| < 1$ but close to 1.0, convergence is quite slow. For the example presented in Table 3.5, $g'(\alpha) = 9.541086$, which explains why that form of $\phi = g(\phi)$ diverges. For the example presented in Table 3.6, $g'(\alpha) = 0.104810$, which means that the error decreases by a factor of approximately 10 at each iteration. Such rapid convergence does not occur when $|g'(\alpha)|$ is close to 1.0. For example, for $|g'(\alpha)| = 0.9$, approximately 22 times as many iterations would be required to reach the same convergence criterion.

If the nonlinear equation, $f(\phi) = 0$, is rearranged into the form $\phi = \phi + f(\phi) = g(\phi)$, the fixed-point iteration formula becomes

$$\phi_{i+1} = \phi_i + f(\phi_i) = g(\phi_i) \quad (3.51)$$

and $g'(\phi)$ is given by

$$g'(\phi) = 1 + R_2 \sin(\phi) - \sin(\alpha - \phi) \quad (3.52)$$

Substituting the final solution value, $\phi = 32.015180$ deg, into Eq. (3.52) gives $g'(\phi) = 2.186449$, which is larger than 1.0. The iteration method would not converge to the desired solution for this rearrangement of $f(\phi) = 0$ into $\phi = g(\phi)$. In fact, the solution converges to $\phi = -9.747105$ deg, for which $g'(\phi) = -0.186449$. This is also a solution to the four-bar linkage problem, but not the desired solution.

Methods which sometimes work and sometimes fail are undesirable. Consequently, the fixed-point iteration method for solving nonlinear equations is not recommended.

Methods for accelerating the convergence of iterative methods based on a knowledge of the convergence rate can be developed. *Aitkens Δ^2 acceleration method* applies to linearly converging methods, such as fixed-point iteration, in which $e_{i+1} = ke_i$. The method is based on starting with an initial approximation x_i for which

$$x_i = \alpha + e_i \quad (3.53a)$$

Two more iterations are made to give

$$x_{i+1} = \alpha + e_{i+1} = \alpha + ke_i \quad (3.53b)$$

$$x_{i+2} = \alpha + e_{i+2} = \alpha + ke_{i+1} = \alpha + k^2 e_i \quad (3.53c)$$

There are three unknowns in Eqs. (3.53a) to (3.53c): e_i , α , and k . These three equations can be solved for these three unknowns. The value of α obtained by the procedure is not the exact root, since higher-order terms have been neglected. However, it is an improved approximation of the root. The procedure is then repeated using α as the initial approximation. It can be shown that the successive approximations to the root, α , converge quadratically. When applied to the fixed-point iteration method for finding the roots of a nonlinear equation, this procedure is known as *Steffensen's method*. Steffensen's method is not developed in this book since Newton's method, which is presented in Section 3.4.2, is a more straightforward procedure for achieving quadratic convergence.

3.4.2. Newton's Method

Newton's method (sometimes called the *Newton-Raphson method*) for solving nonlinear equations is one of the most well-known and powerful procedures in all of numerical analysis. It always converges if the initial approximation is sufficiently close to the root, and it converges quadratically. Its only disadvantage is that the derivative $f'(x)$ of the nonlinear function $f(x)$ must be evaluated.

Newton's method is illustrated graphically in Figure 3.10. The function $f(x)$ is nonlinear. Let's locally approximate $f(x)$ by the linear function $g(x)$, which is tangent to $f(x)$, and find the solution for $g(x) = 0$. Newton's method is sometimes called the tangent method. That solution is then taken as the next approximation to the solution of $f(x) = 0$. The procedure is applied iteratively to convergence. Thus,

$$f'(x_i) = \text{slope of } f(x) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \quad (3.54)$$

Solving Eq. (3.54) for x_{i+1} with $f(x_{i+1}) = 0$ yields

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

(3.55)

Equation (3.55) is applied repetitively until either one or both of the following convergence criteria are satisfied:

$$|x_{i+1} - x_i| \leq \varepsilon_1 \quad \text{and/or} \quad |f(x_{i+1})| \leq \varepsilon_2 \quad (3.56)$$

Newton's method also can be obtained from the Taylor series. Thus,

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + \dots \quad (3.57)$$

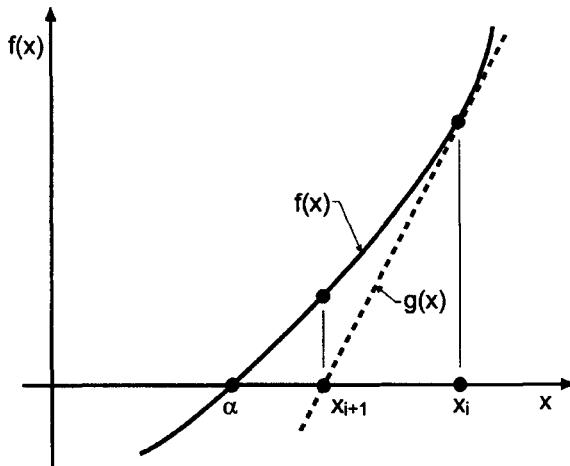


Figure 3.10 Newton's method.

Truncating Eq. (3.57) after the first derivative term, setting $f(x_{i+1}) = 0$, and solving for x_{i+1} yields

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (3.58)$$

Equation (3.58) is the same as Eq. (3.55).

Example 3.4. Newton's method.

To illustrate Newton's method, let's solve the four-bar linkage problem presented in Section 3.1. Recall Eq. (3.3):

$$f(\phi) = R_1 \cos(\alpha) - R_2 \cos(\phi) + R_3 - \cos(\alpha - \phi) = 0 \quad (3.59)$$

The derivative of $f(\phi), f'(\phi)$ is

$$f'(\phi) = R_2 \sin(\phi) - \sin(\alpha - \phi) \quad (3.60)$$

Thus, Eq. (3.55) becomes

$$\phi_{i+1} = \phi_i - \frac{f(\phi_i)}{f'(\phi_i)} \quad (3.61)$$

For $R_1 = \frac{5}{3}$, $R_2 = \frac{5}{2}$, $R_3 = \frac{11}{6}$, and $\alpha = 40.0$ deg, Eqs. (3.59) and (3.60) yield

$$f(\phi) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(\phi) + \frac{11}{6} - \cos(40.0 - \phi) \quad (3.62)$$

$$f'(\phi) = \frac{5}{2} \sin(\phi) - \sin(40.0 - \phi) \quad (3.63)$$

For the first iteration let $\phi_1 = 30.0$ deg. Equations (3.62) and (3.63) give

$$f(\phi_1) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(30.0) + \frac{11}{6} - \cos(40.0 - 30.0) = -0.03979719 \quad (3.64)$$

$$f'(\phi_1) = \frac{5}{2} \sin(30.0) - \sin(40.0 - 30.0) = 1.07635182 \quad (3.65)$$

Substituting these results into Eq. (3.61) yields

$$\phi_2 = 30.0 - \frac{(-0.03979719)(180/\pi)}{1.07635182} = 32.118463 \text{ deg} \quad (3.66)$$

Substituting $\phi_2 = 32.118463$ deg into Eq. (3.62) gives $f(\phi_2) = 0.00214376$.

These results and the results of subsequent iterations are presented in Table 3.7. The convergence criterion, $|\phi_{i+1} - \phi_i| \leq 0.000001$ deg, is satisfied on the fourth iteration. This is a considerable improvement over the interval-halving method, the false position method, and the fixed-point iteration method.

Convergence of Newton's method is determined as follows. Recall Eq. (3.55):

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (3.67)$$

Table 3.7. Newton's Method

i	ϕ_i , deg	$f(\phi_i)$	$f'(\phi_i)$	ϕ_{i+1} , deg	$f(\phi_{i+1})$
1	30.000000	-0.03979719	1.07635164	32.118463	0.00214376
2	32.118463	0.00214376	1.19205359	32.015423	0.00000503
3	32.015423	0.00000503	1.18646209	32.015180	0.00000000
4	32.015180	0.00000000	1.18644892	32.015180	0.00000000
	32.015180	0.00000000			

Equation (3.67) is of the form

$$x_{i+1} = g(x_i) \quad (3.68)$$

where $g(x)$ is given by

$$g(x) = x - \frac{f(x)}{f'(x)} \quad (3.69)$$

As shown by Eq. (3.50), for convergence of any iterative method in the form of Eq. (3.68),

$$|g'(\xi)| \leq 1 \quad (3.70)$$

where ξ lies between x_i and α . From Eq. (3.69),

$$g'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2} \quad (3.71)$$

At the root, $x = \alpha$ and $f(\alpha) = 0$. Thus, $g'(\alpha) = 0$. Consequently, Eq.(3.70) is satisfied, and Newton's method is convergent.

The convergence rate of Newton's method is determined as follows. Subtract α from both sides of Eq. (3.67), and let $e = x - \alpha$ denote the error. Thus,

$$x_{i+1} - \alpha = e_{i+1} = x_i - \alpha - \frac{f(x_i)}{f'(x_i)} = e_i - \frac{f(x_i)}{f'(x_i)} \quad (3.72)$$

Expressing $f(x)$ in a Taylor series about x_i , truncating after the second-order term, and evaluating at $x = \alpha$ yields

$$f(\alpha) = f(x_i) + f'(x_i)(\alpha - x_i) + \frac{1}{2}f''(\xi)(\alpha - x_i)^2 = 0 \quad x_i \leq \xi \leq \alpha \quad (3.73)$$

Letting $e_i = x_i - \alpha$ and solving Eq. (3.73) for $f(x_i)$ gives

$$f(x_i) = f'(x_i)e_i - \frac{1}{2}f''(\xi)e_i^2 \quad (3.74)$$

Substituting Eq. (3.74) into Eq. (3.72) gives

$$e_{i+1} = e_i - \frac{f'(x_i)e_i - \frac{1}{2}f''(\xi)e_i^2}{f'(x_i)} = \frac{1}{2}\frac{f''(\xi)}{f'(x_i)}e_i^2 \quad (3.75)$$

In the limit as $i \rightarrow \infty$, $x_i \rightarrow \alpha$, $f'(x_i) \rightarrow f'(\alpha)$, $f''(\xi) \rightarrow f''(\alpha)$, and Eq. (3.75) becomes

$$e_{i+1} = \frac{1}{2}\frac{f''(\alpha)}{f'(\alpha)}e_i^2 \quad (3.76)$$

Equation (3.76) shows that convergence is second-order, or quadratic. The number of significant figures essentially doubles each iteration.

As the solution is approached, $f(x_i) \rightarrow 0$, and Eq. (3.70) is satisfied. For a poor initial estimate, however, Eq. (3.70) may not be satisfied. In that case, the procedure may converge to an alternate solution, or the solution may jump around wildly for a while and then converge to the desired solution or an alternate solution. The procedure will not diverge disastrously like the fixed-point iteration method when $|g'(x)| > 1$. Newton's method has excellent local convergence properties. However, its global convergence properties can be very poor, due to the neglect of the higher-order terms in the Taylor series presented in Eq. (3.57).

Newton's method requires the value of the derivative $f'(x)$ in addition to the value of the function $f(x)$. When the function $f(x)$ is an algebraic function or a transcendental function, $f'(x)$ can be determined analytically. However, when the function $f(x)$ is a general nonlinear relationship between an input x and an output $f(x)$, $f'(x)$ cannot be determined analytically. In that case, $f'(x)$ can be estimated numerically by evaluating $f(x)$ at x_i and $x_i + \varepsilon$, and approximating $f'(x_i)$ as

$$f'(x_i) = \frac{f(x_i + \varepsilon) - f(x_i)}{\varepsilon} \quad (3.77)$$

This procedure doubles the number of function evaluations at each iteration. However, it eliminates the evaluation of $f'(x)$ at each iteration. If ε is small, round-off errors are introduced, and if ε is too large, the convergence rate is decreased. This process is called the *approximate Newton method*.

In some cases, the efficiency of Newton's method can be increased by using the same value of $f'(x)$ for several iterations. As long as the sign of $f'(x)$ does not change, the iterates x_i move toward the root, $x = \alpha$. However, the second-order convergence is lost, so the overall procedure converges more slowly. However, in problems where evaluation of $f'(x)$ is more costly than evaluation of $f(x)$, this procedure may be less work. This is especially true in the solution of systems of nonlinear equations, which is discussed in Section 3.7. This procedure is called the *lagged Newton's method*.

A higher-order version of Newton's method can be obtained by retaining the second derivative term in the Taylor series presented in Eq. (3.57). This procedure requires the evaluation of $f''(x)$ and the solution of a quadratic equation for $\Delta x = x_{i+1} - x_i$. This procedure is not used very often.

Newton's method can be used to determine complex roots of real equations or complex roots of complex equations simply by using complex arithmetic. Newton's method also can be used to find multiple roots of nonlinear equations. Both of these applications of Newton's method, complex roots and multiple roots, are discussed in Section 3.5, which is concerned with polynomials, which can have both complex roots and multiple roots. Newton's method is also an excellent method for *polishing* roots obtained by other methods which yield results polluted by round-off errors, such as roots of deflated functions (see Section 3.5.2.2).

Newton's method has several disadvantages. Some functions are difficult to differentiate analytically, and some functions cannot be differentiated analytically at all. In such cases, the approximate Newton method defined by Eq. (3.77) or the secant method presented in Section 3.4.3 is recommended. When multiple roots occur, convergence drops to first order. This problem is discussed in Section 3.5.2 for polynomials. The presence of a

local extremum (i.e., maximum or minimum) in $f(x)$ in the neighborhood of a root may cause oscillations in the solution. The presence of inflection points in $f(x)$ in the neighborhood of a root can cause problems. These last two situations are discussed in Section 3.6.1, which is concerned with pitfalls in root finding.

When Newton's method misbehaves, it may be necessary to bracket the solution within a closed interval and ensure that successive approximations remain within the interval. In extremely difficult cases, it may be necessary to make several iterations with the interval halving method to reduce the size of the interval before continuing with Newton's method.

3.4.3. The Secant Method

When the derivative function, $f'(x)$, is unavailable or prohibitively costly to evaluate, an alternative to Newton's method is required. The preferred alternative is the secant method.

The secant method is illustrated graphically in Figure 3.11. The nonlinear function $f(x)$ is approximated locally by the linear function $g(x)$, which is the secant to $f(x)$, and the root of $g(x)$ is taken as an improved approximation to the root of the nonlinear function $f(x)$. A *secant* to a curve is the straight line which passes through two points on the curve. The procedure is applied repetitively to convergence. Two initial approximations, x_0 and x_1 , which are not required to bracket the root, are required to initiate the secant method. The slope of the secant passing through two points, x_{i-1} and x_i , is given by

$$g'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \quad (3.78)$$

The equation of the secant line is given by

$$\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = g'(x_i) \quad (3.79)$$

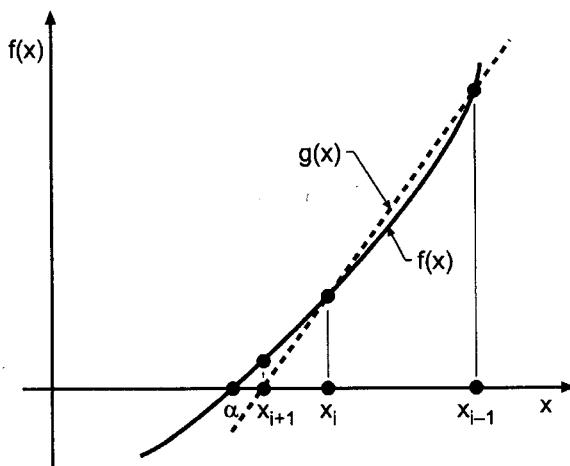


Figure 3.11 The secant method.

where $f(x_{i+1}) = 0$. Solving Eq. (3.79) for x_{i+1} yields

$$x_{i+1} = x_i - \frac{f(x_i)}{g'(x_i)} \quad (3.80)$$

Equation (3.80) is applied repetitively until either one or both of the following two convergence criteria are satisfied:

$$|x_{i+1} - x_i| \leq \varepsilon_1 \quad \text{and/or} \quad |f(x_{i+1})| \leq \varepsilon_2 \quad (3.81)$$

Example 3.5. The secant method.

Let's solve the four-bar linkage problem presented in Section 3.1 by the secant method. Recall Eq. (3.3):

$$f(\phi) = R_1 \cos(\alpha) - R_2 \cos(\phi) + R_3 - \cos(\alpha - \phi) = 0 \quad (3.82)$$

Thus, Eq. (3.80) becomes

$$\phi_{i+1} = \phi_i - \frac{f(\phi_i)}{g'(\phi_i)} \quad (3.83)$$

where $g'(\phi_i)$ is given by

$$g'(\phi_i) = \frac{f(\phi_i) - f(\phi_{i-1})}{\phi_i - \phi_{i-1}} \quad (3.84)$$

For $R_1 = \frac{5}{3}$, $R_2 = \frac{5}{2}$, $R_3 = \frac{11}{6}$, and $\alpha = 40.0 \text{ deg}$, Eq. (3.82) yields

$$f(\phi) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(\phi) + \frac{11}{6} - \cos(40.0 - \phi) \quad (3.85)$$

For the first iteration, let $\phi_0 = 30.0 \text{ deg}$ and $\phi_1 = 40.0 \text{ deg}$. Equation (3.85) gives

$$f(\phi_0) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(30.0) + \frac{11}{6} - \cos(40.0 - 30.0) = -0.03979719 \quad (3.86a)$$

$$f(\phi_1) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(40.0) + \frac{11}{6} - \cos(40.0 - 40.0) = 0.19496296 \quad (3.86b)$$

Substituting these results into Eq. (3.84) gives

$$g'(\phi_1) = \frac{(0.19496296) - (-0.03979719)}{40.0 - 30.0} = 0.02347602 \quad (3.87)$$

Substituting $g'(\phi_1)$ into Eq. (3.83) yields

$$\phi_2 = 40.0 - \frac{0.19496296}{0.02347602} = 31.695228 \text{ deg} \quad (3.88)$$

Substituting $\phi_2 = 31.695228 \text{ deg}$ into Eq. (3.85) gives $f(\phi_2) = -0.00657688$.

These results and the results of subsequent iterations are presented in Table 3.8. The convergence criterion, $|\phi_{i+1} - \phi_i| \leq 0.000001 \text{ deg}$, is satisfied on the fifth iteration, which is one iteration more than Newton's method requires.

Table 3.8. The Secant Method

<i>i</i>	ϕ_i , deg	$f(\phi_i)$	$g'(\phi_i)$	ϕ_{i+1} , deg	$f(\phi_{i+1})$
0	30.000000	-0.03979719			
1	40.000000	0.19496296	0.02347602	31.695228	-0.00657688
2	31.695228	-0.00657688	0.02426795	31.966238	-0.00101233
3	31.966238	-0.00101233	0.02053257	32.015542	0.00000749
4	32.015542	0.00000749	0.02068443	32.015180	-0.00000001
5	32.015180	-0.00000001	0.02070761	32.015180	0.00000000
	32.015180	0.00000000			

The convergence rate of the secant method was analyzed by Jeeves (1958), who showed that

$$e_{i+1} = \left[\frac{1}{2} \frac{f''(\alpha)}{f'(\alpha)} \right]^{0.62...} e_i^{1.62...} \quad (3.89)$$

Convergence occurs at the rate $1.62\dots$, which is considerably faster than the linear convergence rate of the fixed-point iteration method but somewhat slower than the quadratic convergence rate of Newton's method.

The question of which method is more efficient, Newton's method or the secant method, was also answered by Jeeves. He showed that if the effort required to evaluate $f'(x)$ is less than 43 percent of the effort required to evaluate $f(x)$, then Newton's method is more efficient. Otherwise, the secant method is more efficient.

The problems with Newton's method discussed at the end of Section 3.4.2 also apply to the secant method.

3.4.4. Muller's Method

Muller's method (1956) is based on locally approximating the nonlinear function $f(x)$ by a quadratic function $g(x)$, and the root of the quadratic function $g(x)$ is taken as an improved approximation to the root of the nonlinear function $f(x)$. The procedure is applied repetitively to convergence. Three initial approximations x_1 , x_2 , and x_3 , which are not required to bracket the root, are required to start the algorithm. The only difference between Muller's method and the secant method is that $g(x)$ is a quadratic function in Muller's method and a linear function in the secant method.

Muller's method is illustrated graphically in Figure 3.12. The quadratic function $g(x)$ is specified as follows:

$$g(x) = a(x - x_i)^2 + b(x - x_i) + c \quad (3.90)$$

The coefficients (i.e., a , b , and c) are determined by requiring $g(x)$ to pass through the three known points (x_i, f_i) , (x_{i-1}, f_{i-1}) , and (x_{i-2}, f_{i-2}) . Thus

$$g(x_i) = f_i = a(x_i - x_i)^2 + b(x_i - x_i) + c = c \quad (3.91a)$$

$$g(x_{i-1}) = f_{i-1} = a(x_{i-1} - x_i)^2 + b(x_{i-1} - x_i) + c \quad (3.91b)$$

$$g(x_{i-2}) = f_{i-2} = a(x_{i-2} - x_i)^2 + b(x_{i-2} - x_i) + c \quad (3.91c)$$

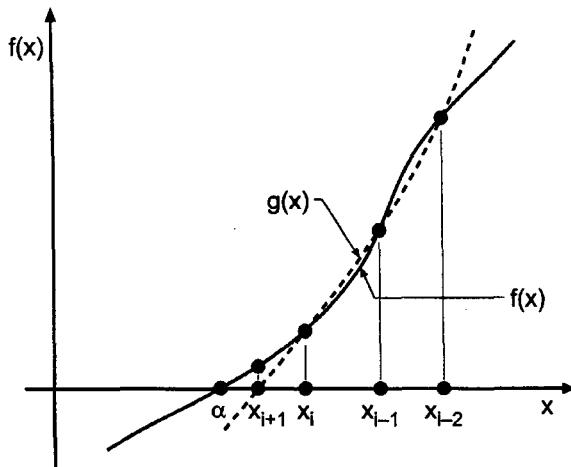


Figure 3.12 Muller's method.

Equation (3.91a) shows that $c = f_i$. Define the following parameters:

$$h_1 = (x_{i-1} - x_i) \quad \text{and} \quad h_2 = (x_{i-2} - x_i) \quad (3.92a)$$

$$\delta_1 = (f_{i-1} - f_i) \quad \text{and} \quad \delta_2 = (f_{i-2} - f_i) \quad (3.92b)$$

Then Eqs. (3.91b) and (3.91c) become

$$h_1^2 a + h_1 b = \delta_1 \quad (3.93a)$$

$$h_2^2 a + h_2 b = \delta_2 \quad (3.93b)$$

Solving Eq. (3.93) by Cramers rule yields

$$a = \frac{\delta_1 h_2 - \delta_2 h_1}{h_1 h_2 (h_1 - h_2)} \quad \text{and} \quad b = \frac{\delta_2 h_1^2 - \delta_1 h_2^2}{h_1 h_2 (h_1 - h_2)} \quad (3.94)$$

Now that the coefficients (i.e., a , b , and c) have been evaluated, Eq. (3.90) can be solved for the value of $(x_{i+1} - x_i)$ which gives $g(x_{i+1}) = 0$. Thus,

$$g(x_{i+1}) = a(x_{i+1} - x_i)^2 + b(x_{i+1} - x_i) + c = 0 \quad (3.95)$$

Solving Eq. (3.95) for $(x_{i+1} - x_i)$ by the rationalized quadratic formula, Eq. (3.112), gives

$$(x_{i+1} - x_i) = -\frac{2c}{b \pm \sqrt{b^2 - 4ac}} \quad (3.96)$$

Solving Eq. (3.96) for x_{i+1} yields

$$x_{i+1} = x_i - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}$$

(3.97)

The sign, + or -, of the square root term in the denominator of Eq. (3.97) is chosen to be the same as the sign of b to keep x_{i+1} close to x_i . Equation (3.97) is applied repetitively until either one or both of the following two convergence criteria are satisfied:

$$|x_{i+1} - x_i| \leq \varepsilon_1 \quad \text{and/or} \quad |f(x_{i+1})| \leq \varepsilon_2 \quad (3.98)$$

Example 3.6. Muller's method.

Let's illustrate Muller's method by solving the four-bar linkage problem presented in Section 3.1. Recall Eq. (3.3):

$$f(\phi) = R_1 \cos(\alpha) - R_2 \cos(\phi) + R_3 - \cos(\alpha - \phi) = 0 \quad (3.99)$$

Thus, Eq. (3.97) becomes

$$\boxed{\phi_{i+1} = \phi_i - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}} \quad (3.100)$$

where $c = f(\phi_i)$ and a and b are given by Eq. (3.94). For $R_1 = \frac{5}{3}$, $R_2 = \frac{5}{2}$, $R_3 = \frac{11}{6}$, and $\alpha = 40.0$ deg. Eq. (3.99) becomes

$$\boxed{f(\phi) = \frac{5}{3} \cos(40.0) - \frac{5}{2} \cos(\phi) + \frac{11}{6} - \cos(40.0 - \phi)} \quad (3.101)$$

For the first iteration, let $\phi_1 = 30.0$ deg, $\phi_2 = 30.5$ deg, and $\phi_3 = 31.0$ deg. Equation (3.101) gives $f(\phi_1) = f_1 = -0.03979719$, $f(\phi_2) = f_2 = -0.03028443$, and $f(\phi_3) = f_3 = -0.02053252$. Thus, $c = f_3 = -0.02053252$. Substituting these values of ϕ_1 , ϕ_2 , f_1 , and f_2 into Eq. (3.92) gives

$$h_1 = (\phi_2 - \phi_3) = -0.50 \quad \text{and} \quad h_2 = (\phi_1 - \phi_3) = -1.00 \quad (3.102a)$$

$$\delta_1 = (f_2 - f_3) = -0.97519103 \quad \text{and} \quad \delta_2 = (f_1 - f_3) = -0.19264670 \quad (3.102b)$$

Substituting these results into Eq. (3.94) yields

$$a = \frac{\delta_1 h_2 - \delta_2 h_1}{h_1 h_2 (h_1 - h_2)} = -0.00047830092 \quad (3.103)$$

$$b = \frac{\delta_2 h_1^2 - \delta_1 h_2^2}{h_1 h_2 (h_1 - h_2)} = 0.019742971 \quad (3.104)$$

Substituting these results into Eq. (3.100) yields

$$\begin{aligned} \phi_{i+1} &= \phi_i - \frac{2c}{b \pm \sqrt{(0.019742971)^2 - 4.0(-0.00047830092)(-0.020532520)}} \\ &= 31.0 - \frac{2(-0.02053252)}{0.019742971 + \sqrt{(0.019742971)^2 - 4.0(-0.00047830092)(-0.020532520)}} \end{aligned} \quad (3.105)$$

which yields $\phi_{i+1} = 32.015031$ deg.

These results and the results of subsequent iterations are presented in Table 3.9. The convergence criterion, $|\phi_{i+1} - \phi_i| \leq 0.000001$ deg, is satisfied on the third iteration.

Table 3.9. Muller's Method

i	ϕ_{i-2} , deg	ϕ_{i-1} , deg	ϕ_i , deg	ϕ_{i+1} , deg	$f(\phi_{i+1})$
1	30.000000				-0.03979717
2	30.500000				-0.03028443
3	31.000000				-0.02053252
4	30.000000	30.500000	31.000000	32.015031	-0.00000309
5	30.500000	31.000000	32.015031	32.015180	0.00000000
6	31.000000	32.015031	32.015180	32.015180	0.00000000
				32.015180	0.00000000

The convergence rate of Muller's method is 1.84, which is faster than the 1.62 rate of the secant method and slower than the 2.0 rate of Newton's method. Generally speaking, the secant method is preferred because of its simplicity, even though its convergence rate, 1.62, is slightly smaller than the convergence rate of Muller's method, 1.84.

3.4.5. Summary

Four open methods for finding the roots of a nonlinear equation are presented in this section: the fixed-point iteration method, Newton's method, the secant method, and Muller's; method. The fixed-point iteration method has a linear convergence rate and converges slowly, or not at all if $|g'(\alpha)| > 1.0$. Consequently, this method is not recommended.

Newton's method, the secant method, and Muller's method all have a higher-order convergence rate (2.0 for Newton's method, 1.62 for the secant method, and 1.84 for Muller's method). All three methods converge rapidly in the vicinity of a root. When the derivative $f'(x)$ is difficult to determine or time consuming to evaluate, the secant method is more efficient. In extremely sensitive problems, all three methods may misbehave and require some bracketing technique. All three of the methods can find complex roots simply by using complex arithmetic. The secant method and Newton's method are highly recommended for finding the roots of nonlinear equations.

3.5 POLYNOMIALS

The methods of solving for the roots of nonlinear equations presented in Sections 3.3 and 3.4 apply to any form of nonlinear equation. One very common form of nonlinear equation is a polynomial. Polynomials arise as the characteristic equation in eigenproblems, in curve-fitting tabular data, as the characteristic equation of higher-order ordinary differential equations, as the characteristic equation in systems of first-order-ordinary differential equations, etc. In all these cases, the roots of the polynomials must be determined. Several special features of solving for the roots of polynomials are discussed in this section.

3.5.1. Introduction

The basic properties of polynomials are presented in Section 4.2. The general form of an n th-degree polynomial is

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \quad (3.106)$$

where n denotes the degree of the polynomial and a_0 to a_n are constant coefficients. The coefficients a_0 to a_n may be real or complex. The evaluation of a polynomial with real coefficients and its derivatives is straightforward, using nested multiplication and synthetic division as discussed in Section 4.2. The evaluation of a polynomial with complex coefficients requires complex arithmetic.

The *fundamental theorem of algebra* states that an n th-degree polynomial has exactly n zeros, or *roots*. The roots may be real or complex. If the coefficients are all real, complex roots always occur in conjugate pairs. The roots may be single (i.e., simple) or repeated (i.e., multiple). The single roots of a linear polynomial can be determined directly. Thus,

$$P_1(x) = ax + b \quad (3.107)$$

has the single root, $x = \alpha$, given by

$$\alpha = -\frac{b}{a} \quad (3.108)$$

The two roots of a second-degree polynomial can also be determined directly. Thus,

$$P_2(x) = ax^2 + bx + c = 0 \quad (3.109)$$

has two roots, α_1 and α_2 , given by the *quadratic formula*:

$$\alpha_1, \alpha_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (3.110)$$

Equation (3.110) yields two distinct real roots when $b^2 > 4ac$, two repeated real roots when $b^2 = 4ac$, and a pair of complex conjugate roots when $b^2 < 4ac$. When $b^2 \gg 4ac$, Eq. (3.110) yields two distinct real roots which are the sum and difference of two nearly identical numbers. In that case, a more accurate result can be obtained by rationalizing Eq. (3.110). Thus,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \left(\frac{-b \mp \sqrt{b^2 - 4ac}}{-b \mp \sqrt{b^2 - 4ac}} \right) \quad (3.111)$$

which yields the *rationalized quadratic formula*:

$$x = -\frac{2c}{b \pm \sqrt{b^2 - 4ac}} \quad (3.112)$$

Exact formulas also exist for the roots of third-degree and fourth-degree polynomials, but they are quite complicated and rarely used. Iterative methods are used to find the roots of higher-degree polynomials.

Descartes' rule of signs, which applies to polynomials having real coefficients, states that the number of positive roots of $P_n(x)$ is equal to the number of sign changes in the nonzero coefficients of $P_n(x)$, or is smaller by an even integer. The number of negative roots is found in a similar manner by considering $P_n(-x)$. For example, the fourth-degree polynomial

$$P_4(x) = -4 + 2x + 3x^2 - 2x^3 + x^4 \quad (3.113)$$

has three sign changes in the coefficients of $P_n(x)$ and one sign change in the coefficients of $P_n(-x) = -4 - 2x + 3x^2 + 2x^3 + x^5$. Thus, the polynomial must have either three positive real roots and one negative real root, or one positive real root, one negative real root, and two complex conjugate roots. The actual roots are $-1, 1, 1 + i\sqrt{10}$, and $1 - i\sqrt{10}$, where $i = \sqrt{-1}$.

The roots of high-degree polynomials can be quite sensitive to small changes in the values of the coefficients. In other words, high-degree polynomials can be *ill-conditioned*. Consider the factored fifth-degree polynomial:

$$P_5(x) = (x - 1)(x - 2)(x - 3)(x - 4)(x - 5) \quad (3.114)$$

which has five positive real roots, 1, 2, 3, 4, and 5. Expanding Eq. (3.114) yields the standard polynomial form:

$$P_5(x) = -120 + 274x - 225x^2 + 85x^3 - 15x^4 + x^5 \quad (3.115)$$

Descartes' rule of signs shows that there are either five positive real roots, or three positive real roots and two complex conjugate roots, or one positive real root and two pairs of complex conjugate roots. To illustrate the sensitivity of the roots to the values of the coefficients, let's change the coefficient of x^2 , which is 225, to 226, which is a change of only 0.44 percent. The five roots are now $1.0514\dots, 1.6191\dots, 5.5075\dots, 3.4110\dots + i1.0793\dots$, and $3.4110\dots - i1.0793\dots$. Thus, a change of only 0.44 percent in one coefficient has made a major change in the roots, including the introduction of two complex conjugate roots. This simple example illustrates the difficulty associated with finding the roots of high-degree polynomials.

One procedure for finding the roots of high-degree polynomials is to find one root by any method, then deflate the polynomial one degree by factoring out the known root using synthetic division, as discussed in Section 4.2. The deflated $(n - 1)$ st-degree polynomial is then solved for the next root. This procedure can be repeated until all the roots are determined. The last two roots should be determined by applying the quadratic formula to the $P_2(x)$ determined after all the deflations. This procedure reduces the work as the subsequent deflated polynomials are of lower and lower degree. It also avoids converging to an already converged root. The major limitation of this approach is that the coefficients of the deflated polynomials are not exact, so the roots of the deflated polynomials are not the precise roots of the original polynomial. Each deflation propagates the errors more and more, so the subsequent roots become less and less accurate. This problem is less serious if the roots are found in order from the smallest to the largest. In general, the roots of the deflated polynomials should be used as first approximations for those roots, which are then refined by solving the original polynomial using the roots of the deflated polynomials as the initial approximations for the refined roots. This process is known as *root polishing*.

The bracketing methods presented in Section 3.3, interval halving and false position, cannot be used to find repeated roots with an even multiplicity, since the nonlinear function $f(x)$ does not change sign at such roots. The first derivative $f'(x)$ does change sign at such roots, but using $f'(x)$ to keep the root bracketed increases the amount of work. Repeated roots with an odd multiplicity can be bracketed by monitoring the sign of $f(x)$, but even in this case the open methods presented in Section 3.4 are more efficient.

Three of the methods presented in Section 3.4 can be used to find the roots of polynomials: Newton's method, the secant method, and Muller's method. Newton's method for polynomials is presented in Section 3.5.2, where it is applied to find a simple root, a multiple root, and a pair of complex conjugate roots.

These three methods also can be used for finding the complex roots of polynomials, provided that complex arithmetic is used and reasonably good complex initial approximations are specified. Complex arithmetic is straightforward on digital computers. However, complex arithmetic is tedious when performed by hand calculation. Several methods exist for extracting complex roots of polynomials that have real coefficients which do not require complex arithmetic. Among these are Bairstow's method, the QD (quotient-difference) method [see Henrici (1964)], and Graeffe's method [see Hildebrand (1956)]. The QD method and Graeffe's method can find all the roots of a polynomial, whereas Bairstow's method extracts quadratic factors which can then be solved by the quadratic formula. Bairstow's method is presented in Section 3.5.3. These three methods use only real arithmetic.

When a polynomial has complex coefficients, Newton's method or the secant method using complex arithmetic and complex initial approximations are the methods of choice.

3.5.2. Newton's method

Newton's method for solving for the root, $x = \alpha$, of a nonlinear equation, $f(x) = 0$, is presented in Section 3.4. Recall Eq. (3.55):

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (3.116)$$

Equation (3.116) will be called Newton's basic method in this section to differentiate it from two variations of Newton's method which are presented in this section for finding multiple roots. Newton's basic method can be used to find simple roots of polynomials, multiple roots of polynomials (where the rate of convergence drops to first order), complex conjugate roots of polynomials with real coefficients, and complex roots of polynomials with complex coefficients. The first three of these applications are illustrated in this section.

3.5.2.1. Newton's Method for Simple Roots.

Newton's basic method can be applied directly to find simple roots of polynomials. Generally speaking, $f(x)$ and $f'(x)$ should be evaluated by the nested multiplication algorithm presented in Section 4.2 for maximum efficiency. No special problems arise. Accurate initial approximations are desirable, and in some cases they are necessary to achieve convergence.

Example 3.7. Newton's method for simple roots.

Let's apply Newton's basic method to find the simple root of the following cubic polynomial in the neighborhood of $x = 1.5$:

$$f(x) = P_3(x) = x^3 - 3x^2 + 4x - 2 = 0 \quad (3.117)$$

Newton's basic method is given by Eq. (3.116). In this example, $f(x_i)$ and $f'(x_i)$ will be evaluated directly for the sake of clarity. The derivative of $f(x)$, $f'(x)$, is given by the second-degree polynomial:

$$f'(x) = P_2(x) = 3x^2 - 6x + 4 \quad (3.118)$$

Table 3.10. Newton's Method for Simple Roots

i	x_i	$f(x_i)$	$f'(x_i)$	x_{i+1}	$f(x_{i+1})$
1	1.50	0.6250	1.750	1.142857	0.14577259
2	1.142857	0.14577259	1.06122449	1.005495	0.00549467
3	1.005495	0.00549476	1.00009057	1.000000	0.000000033
4	1.000000	0.00000033	1.00000000	1.000000	0.000000000
	1.000000	0.00000000			

Let $x_1 = 1.5$. Substituting this value into Eqs. (3.117) and (3.118) gives $f(1.5) = 0.6250$ and $f'(1.5) = 1.750$. Substituting these values into Eq. (3.116) yields

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.5 - \frac{0.6250}{1.750} = 1.142857 \quad (3.119)$$

These results and the results of subsequent iterations are presented in Table 3.10. Four iterations are required to satisfy the convergence tolerance, $|x_{i+1} - x_i| \leq 0.000001$.

Newton's method is an extremely rapid procedure for finding the roots of a polynomial if a reasonable initial approximation is available.

3.5.2.2. Polynomial Deflation

The remaining roots of Eq. (3.117) can be found in a similar manner by choosing different initial approximations. An alternate approach for finding the remaining roots is to *deflate* the original polynomial by factoring out the linear factor corresponding to the known root and solving for the roots of the deflated polynomial.

Example 3.8. Polynomial deflation

Let's illustrate polynomial deflation by factoring out the linear factor, $(x - 1.0)$, from Eq. (3.117). Thus, Eq. (3.117) becomes

$$P_3(x) = (x - 1.0)Q_2(x) \quad (3.120)$$

The coefficients of the deflated polynomial $Q_2(x)$ can be determined by applying the synthetic division algorithm presented in Eq. (4.26). Recall Eq. (3.117):

$$P_3(x) = x^3 - 3x^2 + 4x - 2 \quad (3.121)$$

Applying Eq. (4.26) gives

$$b_3 = a_3 = 1.0 \quad (3.122.3)$$

$$b_2 = a_2 + xb_3 = -3.0 + (1.0)(1.0) = -2.0 \quad (3.122.2)$$

$$b_1 = a_1 + xb_2 = 4.0 + (1.0)(-2.0) = 2.0 \quad (3.122.1)$$

Thus, $Q_2(x)$ is given by

$$x^2 - 2.0x + 2.0 = 0 \quad (3.123)$$

Equation (3.123) is the desired deflated polynomial. Since Eq. (3.123) is a second-degree polynomial, its roots can be determined by the quadratic formula. Thus,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-(-2.0) \pm \sqrt{(-2.0)^2 - 4.0(1.0)(2.0)}}{2(1.0)} \quad (3.124)$$

which yields the complex conjugate roots, $\alpha_{1,2} = 1 \pm i1$.

3.5.2.3. Newton's Method for Multiple Roots

Newton's method, in various forms, can be used to calculate multiple real roots. Ralston and Rabinowitz (1978) show that a nonlinear function $f(x)$ approaches zero faster than its derivative $f'(x)$ approaches zero. Thus, Newton's basic method can be used, but care must be exercised to discontinue the iterations as $f'(x)$ approaches zero. However, the rate of convergence drops to first-order for a multiple root. Two variations of Newton's method restore the second-order convergence of the basic method:

1. Including the multiplicity m in Eq. (3.116)
2. Solving for the root of the modified function, $u(x) = f(x)/f'(x)$

These two variations are presented in the following discussion.

First consider the variation which includes the multiplicity m in Eq. (3.116):

$$x_{i+1} = x_i - m \frac{f(x_i)}{f'(x_{i+1})}$$

(3.125)

Equation (3.125) is in the general iteration form, $x_{i+1} = g(x_i)$. Differentiating $g(x)$ and evaluating the result at $x = \alpha$ yields $g'(\alpha) = 0$. Substituting this result into Eq. (3.50) shows that Eq. (3.125) is convergent. Further analysis yields

$$e_{i+1} = \frac{g''(\xi)}{2} e_i^2 \quad (3.126)$$

where ξ is between x_i and α , which shows that Eq. (3.125) converges quadratically.

Next consider the variation where Newton's basic method is applied to the function $u(x)$:

$$u(x) = \frac{f(x)}{f'(x)} \quad (3.127)$$

If $f(x)$ has m repeated roots, $f(x)$ can be expressed as

$$f(x) = (x - \alpha)^m h(x) \quad (3.128)$$

where the deflated function $h(x)$ does not have a root at $x = \alpha$, that is, $h(\alpha) \neq 0$. Substituting Eq. (3.128) into Eq. (3.127) gives

$$u(x) = \frac{(x - r)^m h(x)}{m(x - \alpha)^{m-1} h(x) + (x - \alpha)^m g'(x)} \quad (3.129)$$

which yields

$$u(x) = \frac{(x - \alpha)h(x)}{mh(x) + (x - \alpha)g'(x)} \quad (3.130)$$

Equation (3.130) shows that $u(x)$ has a single root at $x = \alpha$. Thus, Newton's basic method, with second-order convergence, can be applied to $u(x)$ to give

$$x_{i+1} = x_i - \frac{u(x_i)}{u'(x_i)} \quad (3.131)$$

Differentiating Eq. (3.127) gives

$$u'(x) = \frac{f'(x)f'(x) - f(x)f'(x)}{[f'(x)]^2} \quad (3.132)$$

Substituting Eqs. (3.127) and (3.132) into Eq. (3.131) yields an alternate form of Eq. (3.131):

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)} \quad (3.133)$$

The advantage of Eq. (3.133) over Newton's basic method for repeated roots is that Eq. (3.133) has second-order convergence. There are several disadvantages. There is an additional calculation for $f''(x_i)$. Equation (3.133) requires additional effort to evaluate. Round-off errors may be introduced due to the difference appearing in the denominator of Eq. (3.133). This method can also be used for simple roots, but it is less efficient than Newton's basic method in that case.

In summary, three methods are presented for evaluating repeated roots: Newton's basic method (which reduces to first-order convergence), Newton's basic method with the multiplicity m included, and Newton's basic method applied to the modified function, $u(x) = f(x)/f'(x)$. These three methods can be applied to any nonlinear equation. They are presented in this section devoted to polynomials simply because the problem of multiple roots generally occurs more frequently for polynomials than for other nonlinear functions. The three techniques presented here can also be applied with the secant method, although the evaluation of $f''(x)$ is more complicated in that case. These three methods are illustrated in Example 3.9.

Example 3.9. Newton's method for multiple roots.

Three versions of Newton's method for multiple roots are illustrated in this section:

1. Newton's basic method.
2. Newton's basic method including the multiplicity m .
3. Newton's basic method applied to the modified function, $u(x) = f(x)/f'(x)$.

These three methods are specified in Eqs. (3.116), (3.125), and (3.133), respectively, which are repeated below:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (3.134)$$

$$x_{i+1} = x_i - m \frac{f(x_i)}{f'(x_i)} \quad (3.135)$$

where m is the multiplicity of the root, and

$$x_{i+1} = x_i - \frac{u(x_i)}{u'(x_i)} = x_i - \frac{f(x_i)f'(x_i)}{[f'(x_i)]^2 - f(x_i)f''(x_i)} \quad (3.136)$$

where $u(x) = f(x)/f'(x)$ has the same roots as $f(x)$. Let's solve for the repeated root, $r = 1, 1$, of the following third-degree polynomial:

$$f(x) = P_3(x) = (x + 1)(x - 1)(x - 1) = 0 \quad (3.137)$$

$$f(x) = x^3 - x^2 - x + 1 = 0 \quad (3.138)$$

From Eq. (3.138),

$$f'(x) = 3x^2 - 2x - 1 \quad (3.139)$$

$$f''(x) = 6x - 2 \quad (3.140)$$

Let the initial approximation be $x_1 = 1.50$. From Eqs. (3.138) to (3.140), $f(1.50) = 0.6250$, $f'(1.50) = 2.750$, and $f''(1.5) = 7.0$. Substituting these values into Eqs. (3.134) to (3.136) gives

$$x_2 = 1.5 - \frac{0.6250}{2.750} = 2.272727 \quad (3.141)$$

$$x_2 = 1.5 - 2.0 \frac{0.6250}{2.750} = 1.045455 \quad (3.142)$$

$$x_2 = 1.5 - \frac{(0.6250)(2.750)}{(2.750)^2 - (0.6250)(7.0)} = 0.960784 \quad (3.143)$$

These results and the results of subsequent iterations required to achieve the convergence tolerance, $|\Delta x_{i+1}| \leq 0.000001$, are summarized in Table 3.11.

Newton's basic method required 20 iterations, while the two other methods required only four iterations each. The advantage of these two methods over the basic method for repeated roots is obvious.

3.5.2.4. Newton's Method for Complex Roots

Newton's method, the secant method, and Muller's method can be used to calculate complex roots simply by using complex arithmetic and choosing complex initial approximations.

Bracketing methods, such as interval halving and false position, cannot be used to find complex roots, since the sign of $f(x)$ generally does not change sign at a complex root. Newton's method is applied in this section to find the complex conjugate roots of a polynomial with real coefficients.

Example 3.10. Newton's method for complex roots.

The basic Newton method can find complex roots by using complex arithmetic and choosing a complex initial approximation. Consider the third-degree polynomial:

$$f(x) = P_3(x) = (x - 1)(x - 1 - i1)(x - 1 + i1) \quad (3.144)$$

$$f(x) = x^3 - 3x^2 + 4x - 2 = 0 \quad (3.145)$$

Table 3.11. Newton's Method for Multiple Real Roots

Newton's basic method, Eq. (3.134)

i	x_i	$f(x_i)$	x_{i+1}	$f(x_{i+1})$
1	1.50	0.6250	1.272727	0.16904583
2	1.272727	0.16904583	1.144082	0.04451055
3	1.144082	0.04451055	1.074383	0.01147723
...			
19	1.000002	0.00000000	1.000001	0.00000000
20	1.000000	0.00000000	1.000001	0.00000000
	1.000001	0.00000000		

Newton's multiplicity method, Eq. (3.135), with $m = 2$

i	x_i	$f(x_i)$	x_{i+1}	$f(x_{i+1})$
1	1.50	0.6250	1.045455	0.00422615
2	1.045455	0.00422615	1.00500	0.00000050
3	1.005000	0.00000050	1.000000	0.00000000
4	1.000000	0.00000000	1.000000	0.00000000
	1.000000	0.00000000		

Newton's modified method, Eq. (3.136)

i	x_i	$f(x_i)$	x_{i+1}	$f(x_{i+1})$
1	1.50	0.6250	0.960784	0.00301543
2	0.960784	0.00301543	0.999600	0.00000032
3	0.999600	0.00000032	1.000000	0.00000000
4	1.000000	0.00000000	1.000000	0.00000000
	1.000000	0.00000000		

Table 3.12. Newton's Method for Complex Roots

i	x_i	$f(x_i)$	$f'(x_i)$
1	0.500000 + I0.500000	1.75000000 - I0.25000000	-1.00000000 + I0.50000000
2	2.000000 + I1.000000	1.00000000 + I7.00000000	5.00000000 + I10.00000000
3	1.400000 + I0.800000	0.73600000 + I1.95200000	1.16000000 + I5.12000000
4	1.006386 + I0.854572	0.53189072 + I0.25241794	-1.16521249 - I3.45103149
5	0.987442 + I1.015093	-0.03425358 - I0.08138309	-2.14100172 - I3.98388821
6	0.999707 + I0.999904	0.00097047 - I0.00097901	-2.00059447 - I3.99785801
7	1.000000 + I1.000000	-0.00000002 + I0.00000034	-1.99999953 - I4.00000030
	1.000000 + I1.000000	0.00000000 + I0.00000000	

The roots of Eq. (3.144) are $r = 1, 1 + I1$, and $1 - I1$. Let's find the complex root $r = 1 + I1$ starting with $x_1 = 0.5 + I0.5$. The complex arithmetic was performed by a FORTRAN program for Newton's method. The results are presented in Table 3.12.

3.5.3. Bairstow's Method

A special problem associated with polynomials $P_n(x)$ is the possibility of complex roots. Newton's method, the secant method, and Muller's method all can find complex roots if complex arithmetic is used and complex initial approximations are specified. Fortunately, complex arithmetic is available in several programming languages, such as FORTRAN. However, hand calculation using complex arithmetic is tedious and time consuming. When polynomials with real coefficients have complex roots, they occur in conjugate pairs, which corresponds to a quadratic factor of the polynomial $P_n(x)$. Bairstow's method extracts quadratic factors from a polynomial using only real arithmetic. The quadratic formula can then be used to determine the corresponding pair of real roots or complex conjugate roots.

Consider the general n th-degree polynomial, $P_n(x)$:

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0 \quad (3.146)$$

Let's factor out a quadratic factor from $P_n(x)$. Thus,

$$P_n(x) = (x^2 - rx - s)Q_{n-2}(x) + \text{remainder} \quad (3.147)$$

This form of the quadratic factor (i.e., $x^2 - rx - s$) is generally specified. Performing the division of $P_n(x)$ by the quadratic factor yields

$$P_n(x) = (x^2 - rx - s)(b_n x^{n-2} + b_{n-1} x^{n-3} + \cdots + b_3 x + b_2) + \text{remainder} \quad (3.148)$$

where the remainder is given by

$$\text{Remainder} = b_1(x - r) + b_0 \quad (3.149)$$

When the remainder is zero, $(x^2 - rx - s)$ is an exact factor of $P_n(x)$. The roots of the quadratic factor, real or complex, can be determined by the quadratic formula.

For the remainder to be zero, both b_1 and b_0 must be zero. Both b_1 and b_0 depend on both r and s , thus,

$$b_1 = b_1(r, s) \quad \text{and} \quad b_0 = b_0(r, s) \quad (3.150)$$

Thus, we have a two-variable root-finding problem. This problem can be solved by Newton's method for a system of nonlinear equations, which is presented in Section 3.7.

Expressing Eq. (3.150) in the form of two two-variable Taylor series in terms of $\Delta r = (r^* - r)$ as $\Delta s = (s^* - s)$, where r^* and s^* are the values of r and s which yield $b_1 = b_0 = 0$, gives

$$b_1(r^*, s^*) = b_1 + \frac{\partial b_1}{\partial r} \Delta r + \frac{\partial b_1}{\partial s} \Delta s + \cdots = 0 \quad (3.151a)$$

$$b_0(r^*, s^*) = b_0 + \frac{\partial b_0}{\partial r} \Delta r + \frac{\partial b_0}{\partial s} \Delta s + \cdots = 0 \quad (3.151b)$$

where b_1 , b_0 , and the four partial derivatives are evaluated at point (r, s) . Truncating Eq. (3.151) after the first-order terms and solving for Δr and Δs gives

$$\frac{\partial b_1}{\partial r} \Delta r + \frac{\partial b_1}{\partial s} \Delta s = -b_1 \quad (3.152)$$

$$\frac{\partial b_0}{\partial r} \Delta r + \frac{\partial b_0}{\partial s} \Delta s = -b_0 \quad (3.153)$$

Equations (3.152) and (3.153) can be solved for Δr and Δs by Cramer's rule or Gauss elimination. All that remains is to relate b_1, b_0 , and the four partial derivatives to the coefficients of the polynomial $P_n(x)$, that is, a_i ($i = 0, 1, 2, \dots, n$).

Expanding the right-hand side of Eq. (3.148), including the remainder term, and comparing the two sides term by term, yields

$$b_n = a_n \quad (3.154.n)$$

$$b_{n-1} = a_{n-1} + rb_n \quad (3.154n-1)$$

$$b_{n-2} = a_{n-2} + rb_{n-1} + sb_n \quad (3.154n-2)$$

.....

$$b_1 = a_1 + rb_2 + sb_3 \quad (3.154.1)$$

$$b_0 = a_0 + rb_1 + sb_2 \quad (3.154.0)$$

Equation (3.154) is simply the synthetic division algorithm presented in Section 4.2 applied for a quadratic factor.

The four partial derivatives required in Eqs. (3.152) and (3.153) can be obtained by differentiating the coefficients b_i ($i = n, n - 1, \dots, b_1, b_0$), with respect to r and s , respectively. Since each coefficient b_i contains b_{i+1} and b_{i+2} , we must start with the partial derivatives of b_n and work our way down to the partial derivatives of b_1 and b_0 . Bairstow showed that the results are identical to dividing $Q_{n-2}(x)$ by the quadratic factor, $(x^2 - rx - s)$, using the synthetic division algorithm. The details are presented by Gerald and Wheatley (1999). The results are presented below.

$$c_n = b_n \quad (3.155.n)$$

$$c_{n-1} = b_{n-1} + rc_n \quad (3.155.n-1)$$

$$c_{n-2} = b_{n-2} + rc_{n-1} + sc_n \quad (3.155n-2)$$

.....

$$c_2 = b_2 + rc_3 + sc_4 \quad (3.155.2)$$

$$c_1 = b_1 + rc_2 + sc_3 \quad (3.155.1)$$

The required partial derivatives are given by

$$\frac{\partial b_1}{\partial r} = c_2 \quad \text{and} \quad \frac{\partial b_1}{\partial s} = c_3 \quad (3.156a)$$

$$\frac{\partial b_0}{\partial r} = c_1 \quad \text{and} \quad \frac{\partial b_0}{\partial s} = c_2 \quad (3.156b)$$

Thus, Eqs. (3.152) and (3.153) become

$$c_2 \Delta r + c_3 \Delta s = -b_1 \quad (3.157a)$$

$$c_1 \Delta r + c_2 \Delta s = -b_0 \quad (3.157b)$$

where $\Delta r = (r^* - r)$ and $\Delta s = (s^* - s)$. Thus,

$$r_{i+1} = r_i + \Delta r_i \quad (3.158a)$$

$$s_{i+1} = s_i + \Delta s_i \quad (3.158b)$$

Equations (3.157) and (3.158) are applied repetitively until either one or both of the following convergence criteria are satisfied:

$$|\Delta r_i| \leq \varepsilon_1 \quad \text{and} \quad |\Delta s_i| \leq \varepsilon_1 \quad (3.159a)$$

$$|(b_1)_{i+1} - (b_1)_i| \leq \varepsilon_2 \quad \text{and} \quad |(b_0)_{i+1} - (b_0)_i| \leq \varepsilon_2 \quad (3.159b)$$

Example 3.11. Bairstow's method for quadratic factors.

Let's illustrate Bairstow's method by solving for a quadratic factor of Eq. (3.145):

$$f(x) = x^3 - 3x^2 + 4x - 2 = 0 \quad (3.160)$$

The roots of Eq. (3.160) are $r = 1, 1 + i1$, and $1 - i1$.

To initiate the calculations, let $r_1 = 1.5$ and $s_1 = -2.5$. Substituting these values into Eq. (3.154) gives

$$b_3 = a_3 = 1.0 \quad (3.161.3)$$

$$b_2 = a_2 + rb_3 = (-3.0) + (1.5)(1) = -1.50 \quad (3.161.2)$$

$$b_1 = a_1 + rb_2 + sb_3 = 4.0 + (1.5)(-1.5) + (-2.5)(1.0) = -0.750 \quad (3.161.1)$$

$$b_0 = a_0 + rb_1 + sb_2 = -2.0 + (1.5)(-0.75) + (-2.5)(-1.5) = 0.6250 \quad (3.161.0)$$

Substituting these results into Eq. (3.155) gives

$$c_3 = b_3 = 1.0 \quad (3.162.8)$$

$$c_2 = b_2 + rc_3 = -(1.5) + (1.5)(1.0) = 0.0 \quad (3.162.2)$$

$$c_1 = b_1 + rc_2 + sc_3 = (-0.750) + (1.5)(0.0) + (-2.5)(1.0) = -3.250 \quad (3.162.1)$$

Substituting the values of b_1, b_0, c_3, c_2 , and c_1 into Eq. (3.157) gives

$$(0.0)\Delta r + (1.0)\Delta s = -(-0.75) = 0.750 \quad (3.163a)$$

$$-3.250\Delta r + (0.0)\Delta s = -0.6250 \quad (3.163b)$$

Table 3.13. Bairstow's Method for Quadratic Factors

i	r	s	Δr	Δs
1	1.50	-2.50	0.192308	0.750000
2	1.692308	-1.750	0.278352	-0.144041
3	1.970660	-1.894041	0.034644	-0.110091
4	2.005304	-2.004132	-0.005317	0.004173
5	1.999988	-1.999959	0.000012	-0.000041
6	2.000000	-2.000000	0.000000	0.000000

Solving Eq. (3.163) gives

$$\Delta r = 0.192308 \quad \text{and} \quad \Delta s = 0.750 \quad (3.164)$$

Substituting Δr and Δs into Eq. (3.158) gives

$$r_2 = r_1 + \Delta r = 1.50 + 0.192308 = 1.692308 \quad (3.165a)$$

$$s_2 = s_1 + \Delta s = -2.50 + 0.750 = -1.750 \quad (3.165b)$$

These results and the results of subsequent iterations are presented in Table 3.13. The convergence criteria, $|\Delta r_i| \leq 0.000001$ and $|\Delta s_i| \leq 0.000001$, are satisfied on the sixth iteration, where $r = 2.0$ and $s = -2.0$. Thus, the desired quadratic factor is

$$x^2 - rx - s = x^2 - 2.0x + 2.0 = 0 \quad (3.166)$$

Solving for the roots of Eq. (3.166) by the quadratic formula yields the pair of complex conjugate roots:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-(-2.0) \pm \sqrt{(-2.0)^2 - 4.0(1.0)(2.0)}}{2(1.0)} = 1 + i1, 1 - i1 \quad (3.167)$$

3.5.4. Summary

Polynomials are a special case of nonlinear equation. Any of the methods presented in Sections 3.3 and 3.4 can be used to find the roots of polynomials. Newton's method is especially well suited for this purpose. It can find simple roots and multiple roots directly. However, it drops to first-order for multiple roots. Two variations of Newton's method for multiple roots restore the second-order convergence. Newton's method, like the secant method and Muller's method, can be used to find complex roots simply by using complex arithmetic with complex initial approximations. Bairstow's method can find quadratic factors using real arithmetic, and the quadratic formula can be used to find the two roots of the quadratic factor. Good initial guesses are desirable and may be necessary to find the roots of high-degree polynomials.

3.6 PITFALLS OF ROOT FINDING METHODS AND OTHER METHODS OF ROOT FINDING

The root-finding methods presented in Sections 3.3 to 3.5 generally perform as described. However, there are several pitfalls, or problems, which can arise in their application. Most of these pitfalls are discussed in Sections 3.3 to 3.5. They are summarized and discussed in Section 3.6.1.

The collection of root-finding methods presented in Sections 3.3 to 3.5 includes the more popular methods and the most well-known methods. Several less well-known root-finding methods are listed in Section 3.6.2.

3.6.1. Pitfalls of Root Finding Methods

Numerous pitfalls, or problems, associated with root finding are noted in Sections 3.3 to 3.5. These include:

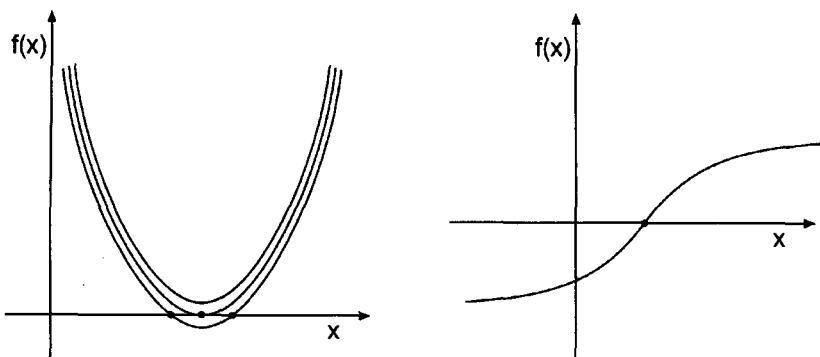


Figure 3.13 Pitfalls of root finding. (a) Closely spaced roots. (b) Inflection point.

1. Lack of a good initial approximation
2. Convergence to the wrong root
3. Closely spaced roots
4. Multiple roots
5. Inflection points
6. Complex roots
7. Ill-conditioning of the nonlinear equation
8. Slow convergence

These problems, and some strategies to avoid the problems, are discussed in this section.

Probably the most serious pitfall associated with root finding is the lack of a good initial approximation. Lack of a good initial approximation can lead to convergence to the wrong root, slow convergence, or divergence. The obvious way to avoid this problem is to obtain a better initial approximation. This can be accomplished by either graphing the function or a fine incremental search.

Closely spaced roots can be difficult to evaluate. Consider the situation illustrated in Figure 3.13. It can be difficult to determine where there are no roots, a double root, or two closely spaced distinct roots. This dilemma can be resolved by an enlargement of a graph of the function or the use of a smaller increment near the root in an incremental search.

Multiple roots, when known to exist, can be evaluated as described for Newton's method in Section 3.5.2. The major problem concerning multiple roots is not knowing they exist. Graphing the function or an incremental search can help identify the possibility of multiple roots.

Roots at an inflection point can send the root-finding procedure far away from the root. A better initial approximation can eliminate this problem.

Complex roots do not present a problem if they are expected. Newton's method or the secant method using complex arithmetic and complex initial approximations can find complex roots in a straightforward manner. However, if complex roots are not expected, and the root-finding method is using real arithmetic, complex roots cannot be evaluated. One solution to this problem is to use Bairstow's method for quadratic factors.

Ill-conditioning of the nonlinear function can cause serious difficulties in root finding. The problems are similar to those discussed in section 1.6.2 for solving ill-

conditioned systems of linear algebraic equations. In root-finding problems, the best approach for finding the roots of ill-conditioned nonlinear equations is to use a computing device with more precision (i.e., a larger number of significant digits).

The problem of slow convergence can be addressed by obtaining a better initial approximation or by a different root-finding method.

Most root-finding problems in engineering and science are well behaved and can be solved in a straightforward manner by one or more of the methods presented in this chapter. Consequently, each problem should be approached with the expectation of success. However, one must always be open to the possibility of unexpected difficulties and be ready and willing to pursue other approaches.

3.6.2. Other Methods of Root Finding

Most of the straightforward popular methods for root finding are presented in Sections 3.3 to 3.5. Several additional methods are identified, but not developed in this section.

Brent's (1978) method uses a superlinear method (i.e., inverse quadratic interpolation) and monitors its behavior to ensure that it is behaving properly. If not, some interval halving steps are used to ensure at least linear behavior until the root is approached more closely, at which time the procedure reverts to the superlinear method. Brent's method does not require evaluation of the derivative. This approach combines the efficiency of open methods with the robustness of closed methods.

Muller's method (1956), mentioned in Section 3.4.4, is an extension of the secant method which approximates the nonlinear function $f(x)$ with a quadratic function $g(x)$, and uses the root of $g(x)$ as the next approximation to the root of $f(x)$. A higher-order version of Newton's method, mentioned in Section 3.4.2, retains the second-order term in the Taylor series for $f(x)$. This method is not used very often because the increased complexity, compared to the secant method and Newton's method, respectively, is not justified by the slightly increased efficiency.

Several additional methods have been proposed for finding the roots of polynomials. Graeff's root squaring method (see Hildebrand, 1956), the Lehmer-Schur method (see Acton, 1970), and the QD (quotient-difference) method (see Henrici, 1964) are three such methods. Two of the more important additional methods for polynomials are Laguerre's method (Householder, 1970) and the Jenkins-Traub method. Ralston and Rabinowitz (1979) present a discussion of these methods. An algorithm for Laguerre's method is presented by Press et al. (1989). The Jenkins-Traub method is implemented in the IMSL library.

3.7 SYSTEMS OF NONLINEAR EQUATIONS

Many problems in engineering and science require the solution of a system of nonlinear equations. Consider a system of two nonlinear equations:

$$\boxed{f(x, y) = 0} \quad (3.168a)$$

$$\boxed{g(x, y) = 0} \quad (3.168b)$$

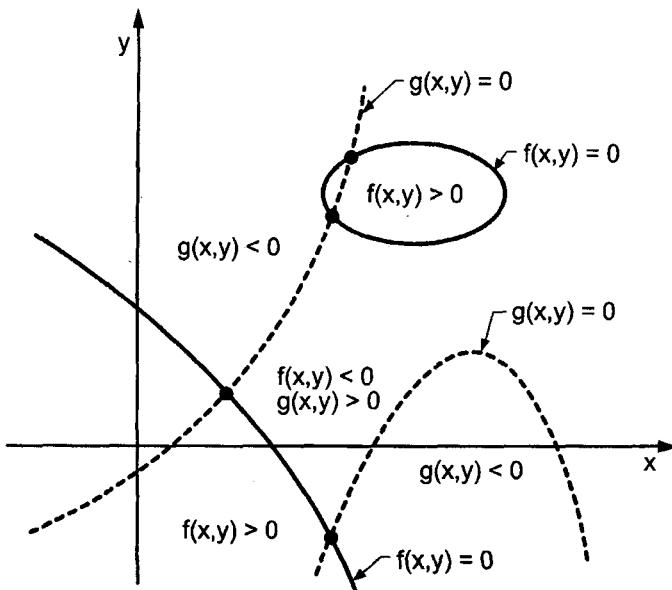


Figure 3.14 Solution of two nonlinear equations.

The problem can be stated as follows:

Given the continuous functions $f(x, y)$ and $g(x, y)$, find the values $x = x^*$ and $y = y^*$ such that $f(x^*, y^*) = 0$ and $g(x^*, y^*) = 0$.

The problem is illustrated graphically in Figure 3.14. The functions $f(x, y)$ and $g(x, y)$ may be algebraic equations, transcendental equations, the solution of differential equations, or any nonlinear relationships between the inputs x and y and the outputs $f(x, y)$ and $g(x, y)$. The $f(x, y) = 0$ and $g(x, y) = 0$ contours divide the xy plane into regions where $f(x, y)$ and $g(x, y)$ are positive or negative. The solutions to Eq. (3.168) are the intersections of the $f(x, y) = g(x, y) = 0$ contours, if any. The number of solutions is not known a priori. Four such intersections are illustrated in Figure 3.14. This problem is considerably more complicated than the solution of a single nonlinear equation.

Interval halving and fixed-point iteration are not readily extendable to systems of nonlinear equations. Newton's method, however, can be extended to solve systems of nonlinear equations. In this section, Newton's method is extended to solve the system of two nonlinear equations specified by Eq. (3.168).

Assume that an approximate solution to Eq. (3.168) is known: (x_i, y_i) . Express $f(x, y)$ and $g(x, y)$ in two-variable Taylor series about (x_i, y_i) , and evaluate the Taylor series at (x^*, y^*) . Thus,

$$f(x^*, y^*) = f_i + f_x|_i(x^* - x_i) + f_y|_i(y^* - y_i) + \dots = 0 \quad (3.169a)$$

$$g(x^*, y^*) = g_i + g_x|_i(x^* - x_i) + g_y|_i(y^* - y_i) + \dots = 0 \quad (3.169b)$$

Truncating Eq. (3.169) after the first derivative terms and rearranging yields

$$f_x|_i \Delta x_i + f_y|_i \Delta y_i = -f_i \quad (3.170a)$$

$$g_x|_i \Delta x_i + g_y|_i \Delta y_i = -g_i \quad (3.170b)$$

where Δx_i and Δy_i denote $(x^* - x_i)$ and $(y^* - y_i)$, respectively. Thus,

$$x_{i+1} = x_i + \Delta x_i \quad (3.171a)$$

$$y_{i+1} = y_i + \Delta y_i \quad (3.171b)$$

Equations (3.170) and (3.171) are applied repetitively until either one or both of the following convergence criteria are satisfied:

$$|\Delta x_i| \leq \varepsilon_x \quad \text{and} \quad |\Delta y_i| \leq \varepsilon_y \quad (3.172a)$$

$$|f(x_{i+1}, y_{i+1})| \leq \varepsilon_f \quad \text{and} \quad |g(x_{i+1}, y_{i+1})| \leq \varepsilon_g \quad (3.172b)$$

Example 3.12. Newton's method for two coupled nonlinear equations.

As an example of Newton's method for solving two nonlinear equations, let's solve the four-bar linkage problem presented in Section 3.1. Recall the two scalar components of the vector loop equation, Eq. (3.2):

$$f(\theta_2, \theta_3) = r_2 \cos(\theta_2) + r_3 \cos(\theta_3) + r_4 \cos(\theta_4) - r_1 = 0 \quad (3.173a)$$

$$g(\theta_2, \theta_3) = r_2 \sin(\theta_2) + r_3 \sin(\theta_3) + r_4 \sin(\theta_4) = 0 \quad (3.173b)$$

where r_1 to r_4 are specified, θ_4 is the input angle, and θ_2 and θ_3 are the two output angles.

Let θ_2^* and θ_3^* be the solution to Eq. (3.173), and θ_2 and θ_3 be an approximation to the solution. Writing Taylor series for $f(\theta_2, \theta_3)$ and $g(\theta_2, \theta_3)$ about (θ_2, θ_3) and evaluating at (θ_2^*, θ_3^*) gives

$$f(\theta_2^*, \theta_3^*) = f|_{\theta_2, \theta_3} + f_{\theta_2}|_{\theta_2, \theta_3} \Delta\theta_2 + f_{\theta_3}|_{\theta_2, \theta_3} \Delta\theta_3 + \dots = 0 \quad (3.174a)$$

$$g(\theta_2^*, \theta_3^*) = g|_{\theta_2, \theta_3} + g_{\theta_2}|_{\theta_2, \theta_3} \Delta\theta_2 + g_{\theta_3}|_{\theta_2, \theta_3} \Delta\theta_3 + \dots = 0 \quad (3.174b)$$

where $\Delta\theta_2 = (\theta_2^* - \theta_2)$ and $\Delta\theta_3 = (\theta_3^* - \theta_3)$. From Eq. (3.173),

$$f_{\theta_2} = -r_2 \sin(\theta_2) \quad \text{and} \quad f_{\theta_3} = -r_3 \sin(\theta_3) \quad (3.175a)$$

$$g_{\theta_2} = r_2 \cos(\theta_2) \quad \text{and} \quad g_{\theta_3} = r_3 \cos(\theta_3) \quad (3.175b)$$

Solving Eqs. (3.174) for $\Delta\theta_2$ and $\Delta\theta_3$ yields the following equations:

$$(f_{\theta_2}|_{\theta_2, \theta_3}) \Delta\theta_2 + (f_{\theta_3}|_{\theta_2, \theta_3}) \Delta\theta_3 = -f(\theta_2, \theta_3) \quad (3.176a)$$

$$(g_{\theta_2}|_{\theta_2, \theta_3}) \Delta\theta_2 + (g_{\theta_3}|_{\theta_2, \theta_3}) \Delta\theta_3 = -g(\theta_2, \theta_3) \quad (3.176b)$$

Equations (3.176a) and (3.176b) can be solved by Cramer's rule or Gauss elimination.

Table 3.14. Newton's Method for Two Coupled Nonlinear Equations

<i>i</i>	θ_2 , deg	θ_3 , deg	$f(\theta_2, \theta_3)$	$g(\theta_2, \theta_3)$	$\Delta\theta_2$, deg	$\Delta\theta_3$, deg
1	30.000000	0.000000	0.131975E + 00	0.428850E + 00	2.520530	-4.708541
2	32.520530	-4.708541	-0.319833E - 01	-0.223639E - 02	-0.500219	0.333480
3	32.020311	-4.375061	-0.328234E - 03	-0.111507E - 03	-0.005130	0.004073
4	32.015181	-4.370988	-0.405454E - 07	-0.112109E - 07	-0.000001	0.000000
	32.015180	-4.370987				

For the problem presented in Section 3.1, $r_1 = 10$, $r_2 = 6$, $r_3 = 8$, and $r_4 = 4$. Consider the case where $\theta_4 = 220.0$ deg. Let $\theta_2^{(1)} = 30.0$ deg and $\theta_3^{(1)} = 0.0$ deg. From Eq. (3.173):

$$f(30.0, 0.0) = 6.0 \cos(30.0) + 8.0 \cos(0.0) + 4.0 \cos(220.0) - 10.0 = 0.131975 \quad (3.177a)$$

$$g(30.0, 0.0) = 6.0 \sin(30.0) + 8.0 \sin(0.0) + 4.0 \sin(220.0) = 0.428850 \quad (3.177b)$$

Equations (3.175a) and (3.175b) give

$$f_{\theta_2} = -6.0 \sin(30.0) = -3.000000 \quad \text{and} \quad f_{\theta_3} = -8.0 \sin(0.0) = 0.0 \quad (3.178a)$$

$$g_{\theta_2} = 6.0 \cos(30.0) = 5.196152 \quad \text{and} \quad g_{\theta_3} = 8.0 \cos(0.0) = 8.0 \quad (3.178b)$$

Substituting these results into Eq. (3.176) gives

$$-3.000000 \Delta\theta_2 + 0.0 \Delta\theta_3 = -0.131975 \quad (3.179a)$$

$$5.196152 \Delta\theta_2 + 8.0 \Delta\theta_3 = -0.428850 \quad (3.179b)$$

Solving Eq. (3.179) gives

$$\Delta\theta_2 = 0.043992(180/\pi) = 2.520530 \text{ deg} \quad (3.180a)$$

$$\Delta\theta_3 = -0.082180(180/\pi) = -4.708541 \text{ deg} \quad (3.180b)$$

where the factor $(180/\pi)$ is needed to convert radians to degrees. Thus,

$$\theta_2 = 32.520530 \text{ deg} \quad \text{and} \quad \theta_3 = -4.708541 \text{ deg} \quad (3.181)$$

These results and the results of subsequent iterations are presented in Table 3.14. Four iterations are required to satisfy the convergence criteria $|\Delta\theta_2| \leq 0.000001$ and $|\theta_3| \leq 0.000001$. These results were obtained on a 13-digit precision computer. As illustrated in Figure 3.1, $\theta_2 = \phi$. From Table 3.1, $\phi = 32.015180$ deg, which is the same as θ_2 .

In the general case,

$$\boxed{\mathbf{f}(\mathbf{x}) = 0} \quad (3.182)$$

where $\mathbf{f}(\mathbf{x})^T = [f_1(\mathbf{x}) \quad f_2(\mathbf{x}) \quad \cdots \quad f_n(\mathbf{x})]$ and $\mathbf{x}^T = [x_1 \quad x_2 \quad \cdots \quad x_n]$. In this case, Eqs. (3.170) and (3.171) become

$$\boxed{\mathbf{A}\Delta = \mathbf{f}} \quad (3.183)$$

where \mathbf{A} is the $n \times n$ matrix of partial derivatives,

$$\mathbf{A} = \begin{bmatrix} (f_1)_{x_1} & (f_1)_{x_2} & \cdots & (f_1)_{x_n} \\ (f_2)_{x_1} & (f_2)_{x_2} & \cdots & (f_2)_{x_n} \\ \dots & \dots & \dots & \dots \\ (f_n)_{x_1} & (f_n)_{x_2} & \cdots & (f_n)_{x_n} \end{bmatrix} \quad (3.184)$$

Δ is the column vector of corrections,

$$\Delta^T = [\Delta x_1 \quad \Delta x_2 \quad \cdots \quad \Delta x_n] \quad (3.185)$$

and \mathbf{f} is the column vector of function values

$$\mathbf{f}^T = [f_1 \quad f_2 \quad \cdots \quad f_n] \quad (3.186)$$

The most costly part of solving systems of nonlinear equations is the evaluation of the matrix of partial derivatives, \mathbf{A} . Letting \mathbf{A} be constant may yield a much less costly solution. However, \mathbf{A} must be reasonably accurate for this procedure to work. A strategy based on making several corrections using constant \mathbf{A} , then reevaluating \mathbf{A} , may yield the most economical solution.

In situations where the partial derivatives of $\mathbf{f}(\mathbf{x})$ cannot be evaluated analytically, the above procedure cannot be applied. One alternate approach is to estimate the partial derivatives in Eq. (3.184) numerically. Thus,

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(\mathbf{x} + \Delta x_j) - f_i(\mathbf{x})}{\Delta x_j} \quad (i, j = 1, 2, \dots, n) \quad (3.187)$$

This procedure has two disadvantages. First, the number of calculations is increased. Second, if Δx_j is too small, round-off errors pollute the solution, and if Δx_j is too large, the convergence rate can decrease to first order. Nevertheless, this is one procedure for solving systems of nonlinear equations where the partial derivatives of $\mathbf{f}(\mathbf{x})$ cannot be determined analytically.

An alternate approach involves constructing a single nonlinear function $F(\mathbf{x})$ by adding together the sums of the squares of the individual functions $f_i(\mathbf{x})$. The nonlinear function $F(\mathbf{x})$ has a global minimum of zero when all of the individual functions are zero. Multidimensional minimization techniques can be applied to minimize $F(\mathbf{x})$, which yields the solution to the system of nonlinear equations, $\mathbf{f}(\mathbf{x}) = 0$. Dennis et al. (1983) discuss such procedures.

3.8 PROGRAMS

Three FORTRAN subroutines for solving nonlinear equations are presented in this section:

1. Newton's method
2. The secant method
3. Newton's method for two simultaneous equations

The basic computational algorithms are presented as completely self-contained subroutines suitable for use in other programs. Input data and output statements are contained in a main (or driver) program written specifically to illustrate the use of each subroutine.

3.8.1. Newton's Method

The general algorithm for Newton's method is given by Eq. (3.55):

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad (3.188)$$

A FORTRAN subroutine, *subroutine newton*, for implementing Newton's method is presented in Program 3.1. *Subroutine newton* requires a function subprogram, *function funct*, which evaluates the nonlinear equation of interest. *Function funct* must be completely self-contained, including all numerical values and conversion factors. The value of x is passed to *function funct*, and the values of $f(x)$ and $f'(x)$ are returned as f and fp , respectively. Every nonlinear equation requires its own individual *function funct*. *Subroutine newton* calls *function funct* to evaluate $f(x)$ and $f'(x)$ for a specified value of x , applies Eq. (3.188), checks for convergence, and either continues or returns. After *iter* iterations, an error message is printed and the solution is terminated. Program 3.1 defines the data set and prints it, calls *subroutine newton* to implement the solution, and prints the solution.

Program 3.1. Newton's method program.

```

program main
c   main program to illustrate nonlinear equation solvers
c   x1   first guess for the root
c   iter  number of iterations allowed
c   tol   convergence tolerance
c   iw    intermediate results output flag: 0 no, 1 yes
      data x1,iter,tol,iw / 30.0, 10, 0.000001, 1 /
      write (6,1000)
      write (6,1010)
      call newton (x1,iter,tol,iw,i)
      call funct (x1,f1,fp1)
      write (6,1020) i,x1,f1
      stop
1000 format (' Newtons method')
1010 format (' //',i',6x,'xi',10x,'fi',12x,'fpi',11x,'xi+1'// ')
1020 format (i4,f12.4,g,f14.8)
      end

      subroutine newton (x1,iter,tol,iw,i)
c   Newton's method
      do i=1,iter
          call funct (x1,f1,fp1)
          dx=-f1/fp1
          x2=x1+dx
          if (iw.eq.1) write (6,1000) i,x1,f1,fp1,x2
          x1=x2
          if (abs(dx).le.tol) return
      end do
      write (6,1010)
      return
1000 format (i4,f12.4,g,2f14.8,f12.6)

```

```

1010 format (' // Iterations failed to converge')
end

function funct (x,f,fp)
c evaluates the nonlinear function
data r1,r2,r3,alpha / 1.66666667, 2.5, 1.83333333, 40.0 /
rad=acos(-1.0)/180.0
f=r1*cos(alpha*rad)-r2*cos(x*rad)+r3-cos((alpha-x)*rad)
fp=(r2*sin(x*rad)-sin((alpha-x)*rad))*rad
return
end

```

The data set used to illustrate *subroutine newton* is taken from Example 3.4. The output generated by the Newton method program is presented in Output 3.1.

Output 3.1. Solution by Newton's method.

Newtons method

i	xi	fi	fpi	xi+1
1	30.000000	-0.03979719	0.01878588	32.118463
2	32.118463	0.00214376	0.02080526	32.015423
3	32.015423	0.00000503	0.02070767	32.015180
4	32.015180	0.00000000	0.02070744	32.015180
4	32.015180	0.00000000		

Subroutine newton can be used to solve most of the nonlinear equations presented in this chapter. The values in the *data* statement must be changed accordingly, and the function subprogram, *function funct*, must evaluate the desired nonlinear equation. Complex roots can be evaluated simply by declaring all variables to be complex variables. As an additional example, *function funct* presented below evaluates a real coefficient polynomial up to fourth degree. This *function funct* is illustrated by solving Example 3.7. Simple roots can be evaluated directly. Multiple roots can be evaluated in three ways: directly (which reduces the order to first order), by including the multiplicity *m* as a coefficient of *f(x)* in *function funct*, or by defining *u(x) = f(x)/f'(x)* in *function funct*.

Polynomial function funct.

```

function funct (x,f,fp)
c evaluates a polynomial of up to fourth degree
data a0,a1,a2,a3,a4 / -2.0, 4.0, -3.0, 1.0, 0.0 /
f=a0+a1*x+a2*x**2+a3*x**3+a4*x**4
fp=a1+2.0*a2*x+3.0*a3*x**2+4.0*a4*x**3
return
end

```

The data set used to illustrate the polynomial *function funct* is taken from Example 3.7. The results are presented below.

Solution by Newton's method.

Newtons method

<i>i</i>	<i>xi</i>	<i>fi</i>	<i>fpi</i>	<i>xi+1</i>
1	1.500000	0.62500000	1.75000000	1.142857
2	1.142857	0.14577259	1.06122449	1.005495
3	1.005495	0.00549467	1.00009057	1.000000
4	1.000000	0.00000033	1.00000000	1.000000
4	1.000000	0.00000000		

3.8.2. The Secant Method

The general algorithm for the secant method is given by Eq. (3.80):

$$x_{i+1} = x_i - \frac{f(x_i)}{g'(x_i)} \quad (3.189)$$

A FORTRAN subroutine, *subroutine secant*, for implementing the secant method is presented below. *Subroutine secant* works essentially like *subroutine newton* discussed in Section 3.8.1, except two values of *x*, *x*₁ and *x*₂, are supplied instead of only one value. Program 3.2 defines the data set and prints it, calls *subroutine secant* to implement the secant method, and prints the solution. Program 3.2 shows only the statements which are different from the statements in Program 3.1.

Program 3.2. The secant method program.

```

program main
c      main program to illustrate nonlinear equation solvers
c      x2      second guess for the root
      data x1,x2,iter,tol,iw / 30.0, 40.0, 10, 0.000001, 1 /
      call secant (x1,x2,iter,tol,iw,i)
      call funct (x1,f1)
1000 format (' The secant method')
1010 format (' /' i',6x,'xi',10x,'fi',12x,'gpi',11x,'xi+1'/' ')
      end

      subroutine secant (x1,x2,iter,tol,iw,i)
c      the secant method
      call funct (x1,f1)
      if (iw.eq.1) write (6,1000) i,x1,f1
      do i=1,iter
          call funct (x2,f2)
          gp2=(f2-f1)/(x2-x1)
          dx=-f2/gp2
          x3=x2+dx

```

```

if (iw.eq.1) write (6,1000) i,x2,f2,gp2,x3
x1=x3
if (abs(dx).le.tol) return
x1=x2
f1=f2
x2=x3
end do
write (6,1010)
return
1000 format (i4,f12.4,g,2f14.8,f12.6)
1010 format (' // Iterations failed to converge')
end

function funct (x,f)
c evaluates the nonlinear function
end

```

The data set used to illustrate *subroutine secant* is taken from Example 3.5. The output generated by the secant method program is presented in Output 3.2.

Output 3.2. Solution by the secant method.

The secant method

i	xi	fi	gpi	xi+1
0	30.000000	-0.03979719		
1	40.000000	0.19496296	0.02347602	31.695228
2	31.695228	-0.00657688	0.02426795	31.966238
3	31.966238	-0.00101233	0.02053257	32.015542
4	32.015542	0.00000749	0.02068443	32.015180
5	32.015180	-0.00000001	0.02070761	32.015180
5	32.015180	0.00000000		

3.8.3. Newton's Method for Two Coupled Nonlinear Equations

The general algorithm for Newton's method for two simultaneous nonlinear equations is given by Eqs. (3.170) and (3.171).

$$f_x|_i \Delta x_i + f_y|_i \Delta y_i = -f_i \quad (3.190a)$$

$$g_x|_i \Delta x_i + g_y|_i \Delta y_i = -g_i \quad (3.190b)$$

$$x_{i+1} = x_i + \Delta x_i \quad (3.191a)$$

$$y_{i+1} = y_i + \Delta y_i \quad (3.191b)$$

The general approach to this problem is the same as the approach presented in Section 3.8.1 for a single nonlinear equation. A FORTRAN subroutine, *subroutine simul*, for implementing the procedure is presented below. Program 3.3 defines the data set and prints it, calls *subroutine simul* to implement the solution, and prints the solution.

Program 3.3. Newton's method for simultaneous equations program.

```

program main
c      main program to illustrate nonlinear equation solvers
c      x1,y1 first guess for the root
c      iter number of iterations allowed
c      tol convergence tolerance
c      iw intermediate results output flag: 0 no, 1 yes
data x1,y1,iter,tol,iw / 30.0, 0.0, 10, 0.000001, 1 /
write (6,1000)
write (6,1010)
call simul (x1,y1,iter,tol,iw,i)
call funct (x1,y1,f1,g1,fx,fy,gx,gy)
write (6,1020) i,x1,y1,f1,g1
stop
1000 format (' Newtons method for two coupled nonlinear equations')
1010 format (' // i',6x,'xi',10x,'yi',9x,'fi',10x,'gi',9x,'dx',
1 6x,'dy'// ' ')
1020 format (i3,2f12.6,2e12.4)
end

subroutine simul (x1,y1,iter,tol,iw,i)
c      Newton's method for two coupled nonlinear equations
do i=1,iter
    call funct (x1,y1,f1,g1,fx,fy,gx,gy)
    del=f1*gy-fy*g1
    dx=(fy*g1-f1*gy)/del
    dy=(f1*gx-fx*g1)/del
    x2=x1+dx
    y2=y1+dy
    if (iw.eq.1) write (6,1000) i,x1,y1,f1,g1,dx,dy
    x1=x2
    y1=y2
    if ((abs(dx).le.tol).and.(abs(dy).le.tol)) return
end do
write (6,1010)
return
1000 format (i3,2f12.6,2e12.4,2f8.4)
1010 format (' // Iteration failed to converge')
end

function funct (x,y,f,g,fx,fy,gx,gy)
c      evaluates the two nonlinear functions
data r1,r2,r3,r4,theta4 / 10.0, 6.0, 8.0, 4.0, 220.0 /
rad=acos(-1.0)/180.0
f=r2*cos(x*rad)+r3*cos(y*rad)+r4*cos(theta4*rad)-r1
g=r2*sin(x*rad)+r3*sin(y*rad)+r4*sin(theta4*rad)
fx=(-r2*sin(x*rad))*rad
fy=(-r3*sin(y*rad))*rad
gx=(r2*cos(x*rad))*rad
gy=(r3*cos(y*rad))*rad
return
end

```

The data set used to illustrate subroutine *simul* is taken from Example 3.12. The output is presented in Output 3.3.

Output 3.3. Solution by Newton's method for simultaneous equations.

Newtons method for two coupled nonlinear equations

i	xi	yi	fi	gi	dx	dy
1	30.000000	0.000000	0.1320E+00	0.4288E+00	2.5205	-4.7085
2	32.520530	-4.708541	-0.3198E-01	-0.2236E-02	-0.5002	0.3335
3	32.020311	-4.375061	-0.3282E-03	-0.1115E-03	-0.0051	0.0041
4	32.015181	-4.370988	-0.4055E-07	-0.1121E-07	0.0000	0.0000
4	32.015180	-4.370987	0.0000E+00	0.0000E+00		

3.8.4. Packages for Nonlinear Equations

Numerous libraries and software packages are available for solving nonlinear equations. Many workstations and mainframe computers have such libraries attached to their operating systems.

Many commercial software packages contain nonlinear equation solvers. Some of the more prominent packages are Matlab and Mathcad. More sophisticated packages, such as IMSL, Mathematica, Macsyma, and Maple, also contain nonlinear equation solvers. Finally, the book *Numerical Recipes* (Press et al., 1989) contains numerous subroutines for solving nonlinear equations.

3.9 SUMMARY

Several methods for solving nonlinear equations are presented in this chapter. The nonlinear equation may be an algebraic equation, a transcendental equation, the solution of a differential equation, or any nonlinear relationship between an input x and a response $f(x)$.

Interval halving (bisection) and false position (regula falsi) converge very slowly, but are certain to converge because the root lies in a closed interval. These methods are not recommended unless the nonlinear equation is so poorly behaved that all other methods fail.

Fixed-point iteration converges only if the derivative of the nonlinear function is less than unity in magnitude. Consequently, it is not recommended.

Newton's method and the secant method are both effective methods for solving nonlinear equations. Both methods generally require reasonable initial approximations. Newton's method converges faster than the secant method (i.e., second order compared to 1.62 order), but Newton's method requires the evaluation of the derivative of the nonlinear function. If the effort required to evaluate the derivative is less than 43 percent of the effort required to evaluate the function itself, Newton's method requires less total effort than the secant method. Otherwise, the secant method requires less total effort. For functions whose derivative cannot be evaluated, the secant method is recommended. Both methods can find complex roots if complex arithmetic is used. The secant method is recommended as the best general purpose method.

The higher-order variations of Newton's method and the secant method, that is, the second-order Taylor series method and Muller's method, respectively, while quite effective, are not used frequently. This is probably because Newton's method and the secant method

are so efficient that the slightly more complicated logic of the higher-order methods is not justified.

Polynomials can be solved by any of the methods for solving nonlinear equations. However, the special features of polynomials should be taken into account.

Multiple roots can be evaluated using Newton's basic method or its variations.

Complex roots can be evaluated by Newton's method or the secant method by using complex arithmetic and complex initial approximations. Complex roots can also be evaluated by Bairstow's method for quadratic factors.

Solving systems of nonlinear equations is a difficult task. For systems of nonlinear equations which have analytical partial derivatives, Newton's method can be used. Otherwise, multidimensional minimization techniques may be preferred. No single approach has proven to be the most effective. Solving systems of nonlinear equations remains a difficult problem.

After studying Chapter 3, you should be able to:

1. Discuss the general features of root finding for nonlinear equations
2. Explain the concept of bounding a root
3. Discuss the benefits of graphing a function
4. Explain how to conduct an incremental search
5. Explain the concept of refining a root
6. Explain the difference between closed domain (bracketing methods) and open domain methods
7. List several closed domain (bracketing) methods
8. List several open domain methods
9. Discuss the types of behavior of nonlinear equations in the neighborhood of a root
10. Discuss the general philosophy of root finding
11. List two closed domain (bracketing) methods
12. Explain how the interval halving (bisection) method works
13. Apply the interval halving (bisection) method
14. List the advantages and disadvantages of the interval halving (bisection) method
15. Explain how the false position (regula falsi) method works
16. Apply the false position (regula falsi) method
17. List the advantages and disadvantages of the false position (regula falsi) method
18. List several open domain methods
19. Explain how the fixed-point iteration method works
20. Apply the fixed-point iteration method
21. List the advantages and disadvantages of the fixed-point iteration method
22. Explain how Newton's method works
23. Apply Newton's method
24. List the advantages and disadvantages of Newton's method
25. Explain and apply the approximate Newton's method
26. Explain and apply the lagged Newton's method
27. Explain how the secant method works
28. Apply the secant method
29. List the advantages and disadvantages of the secant method
30. Explain the lagged secant method

31. Explain how Muller's method works
32. Apply Muller's method
33. List the advantages and disadvantages of Muller's method
34. Discuss the special features of polynomials
35. Apply the quadratic formula and the rationalized quadratic formula
36. Discuss the applicability (or nonapplicability) of closed domain methods and open domain methods for finding the roots of polynomials
37. Discuss the problems associated with finding multiple roots and complex roots
38. Apply Newton's basic method and its variations to find all the roots of a polynomial
39. Apply deflation to a polynomial
40. Explain the concepts underlying Bairstow's method for finding quadratic factors
41. Apply Bairstow's method to find real or complex roots
42. Discuss and give examples of the pitfalls of root finding
43. Suggest ways to get around the pitfalls
44. Explain the concepts underlying Newton's method for a system of nonlinear equations
45. Apply Newton's method to a system of nonlinear equations

EXERCISE PROBLEMS

In all of the problems in this chapter, carry at least six significant figures in all calculations, unless otherwise noted. Continue all iterative procedures until four significant figures have converged, unless otherwise noted. Consider the following four nonlinear equations:

$$\begin{array}{lll} f(x) = x - \cos(x) = 0 & \text{(A)} & f(x) = e^x - \sin(\pi x/3) = 0 \\ f(x) = e^x - 2x - 2 = 0 & \text{(C)} & f(x) = x^3 - 2x^2 - 2x + 1 = 0 \end{array} \quad \text{(B)} \quad \text{(D)}$$

3.2 Closed Domain Methods

Interval Halving

1. Solve Eq. (A) by interval-halving starting with $x = 0.5$ and 1.0 .
2. Solve Eq. (B) by interval-halving starting with $x = -3.5$ and -2.5 .
3. Solve Eq. (C) by interval-halving starting with $x = 1.0$ and 2.0 .
4. Solve Eq. (D) by interval-halving starting with $x = 0.0$ and 1.0 .
5. Find the two points of intersection of the two curves $y = e^x$ and $y = 3x + 2$ using interval halving. Use $(-1.0$ and $0.0)$ and $(2.0$ and $3.0)$ as starting values.
6. Find the two points of intersection of the two curves $y = e^x$ and $y = x^4$ using interval halving. Use $(-1.0$ and $0.0)$ and $(1.0$ and $2.0)$ as starting values.
7. Problems 1 to 6 can be solved using any two initial values of x that bracket the root. Choose other sets of initial values of x to gain additional experience with interval halving.

False Position

8. Solve Eq. (A) by false position starting with $x = 0.5$ and 1.0 .
9. Solve Eq. (B) by false position starting with $x = -3.5$ and -2.5 .
10. Solve Eq. (C) by false position starting with $x = 1.0$ and 2.0 .

11. Solve Eq. (D) by false position starting with $x = 0.0$ and 1.0 .
12. Find the two points of intersection of the two curves $y = e^x$ and $y = 3x + 2$ using false position. Use $(-1.0$ and $0.0)$ and $(2.0$ and $3.0)$ as starting values.
13. Find the two points of intersection of the two curves $y = e^x$ and $y = x^4$ using false position. Use $(-1.0$ and $0.0)$ and $(1.0$ and $2.0)$ as starting values.
14. Problems 8 to 13 can be solved using any two initial values of x that bracket the root. Choose other sets of initial values of x to gain additional experience with false position.

3.4 Open Domain Methods

Fixed-Point Iteration

15. Solve Eq. (A) by fixed-point iteration with $x_0 = 0.5$.
16. Solve Eq. (A) by fixed-point iteration with $x_0 = 1.0$.
17. Solve Eq. (B) by fixed-point iteration with $x_0 = -3.5$.
18. Solve Eq. (B) by fixed-point iteration with $x_0 = -2.5$.
19. Solve Eq. (C) by fixed-point iteration with $x_0 = 1.0$.
20. Solve Eq. (C) by fixed-point iteration with $x_0 = 2.0$.
21. Solve Eq. (D) by fixed-point iteration with $x_0 = 0.0$.
22. Solve Eq. (D) by fixed-point iteration with $x_0 = 1.0$.
23. Problem 5 considers the function $f(x) = e^x - (3x + 2) = 0$, which can be rearranged into the following three forms: (a) $x = e^x - (2x + 2)$, (b) $x = (e^x - 2)/3$, and (c) $x = \ln(3x + 2)$. Solve for the positive root by fixed-point iteration for all three forms, with $x_0 = 1.0$.
24. Solve Problem 6 by fixed-point iteration with $x_0 = -1.0$ and $x_0 = 1.0$.
25. The function $f(x) = (x + 2)(x - 4) = x^2 - 2x - 8 = 0$ has the two roots $x = -2$ and 4 . Rearrange $f(x)$ into the form $x = g(x)$ to obtain the root (a) $x = -2$ and (b) $x = 4$, starting with $x_0 = -1$ and 3 , respectively. The function $f(x)$ can be rearranged in several ways, for example, (a) $x = 8/(x - 2)$, (b) $x = (2x + 8)^{1/2}$, and (c) $x = (x^2 - 8)/2$. One form always converges to $x = -2$, one form always converges to $x = 4$, and one form always diverges. Determine the behavior of the three forms.
26. For what starting values of x might the expression $x = 1/(x + 1)$ not converge?
27. The function $f(x) = e^x - 3x^2 = 0$ has three roots. The function can be rearranged into the form $x = \pm [e^x/3]^{1/2}$. Starting with $x_0 = 0.0$, find the roots corresponding to (a) the $+$ sign (near $x = 1.0$) and (b) the $-$ sign (near -0.5). (c) The third root is near $x = 4.0$. Show that the above form will not converge to this root, even with an initial guess close to the exact root. Develop a form of $x = g(x)$ that will converge to this root, and solve for the third root.
28. The cubic polynomial $f(x) = x^3 + 3x^2 - 2x - 4 = 0$ has a root near $x = 1.0$. Find two forms of $x = g(x)$ that will converge to this root. Solve these two forms for the root, starting with $x_0 = 1.0$.

Newton's Method

29. Solve Eq. (A) by Newton's method. Use $x_0 = 1.0$ as the starting value.
30. Solve Eq. (B) by Newton's method. Use $x_0 = -3.0$ as the starting value.

31. Solve Eq. (C) by Newton's method. Use $x_0 = 1.0$ as the starting value.
32. Solve Eq. (D) by Newton's method. Use $x_0 = 1.0$ as the starting value.
33. Find the positive root of $f(x) = x^{15} - 1 = 0$ by Newton's method, starting with $x_0 = 1.1$.
34. Solve Problem 33 using $x = 0.5$ as the initial guess. You may want to solve this problem on a computer, since a large number of iterations may be required.
35. The n th root of the number N can be found by solving the equation $x^n - N = 0$. (a) For this equation, show that Newton's method gives

$$x_{i+1} = \frac{1}{n} \left[(n-1)x_i + \frac{N}{x_1^{n-1}} \right] \quad (\text{E})$$

Use the above result to solve the following problems: (a) $(161)^{1/3}$, (b) $(21.75)^{1/4}$, (c) $(238.56)^{1/5}$. Use $x = 6.0$, 2.0 , and 3.0 , respectively, as starting values.

36. Consider the function $f(x) = e^x - 2x^2 = 0$. (a) Find the two positive roots using Newton's method. (b) Find the negative root using Newton's method.

The Secant Method

37. Solve Eq. (A) by the secant method. Use $x = 0.5$ and 1.0 as starting values.
38. Solve Eq. (B) by the secant method. Use $x_0 = -3.0$ and -2.5 as starting values.
39. Solve Eq. (C) by the secant method. Use $x_0 = 1.0$ and 2.0 as starting values.
40. Solve Eq. (D) by the secant method. Use $x_0 = 0.0$ and 1.0 as starting values.
41. Find the positive root of $f(x) = x^{15} - 1 = 0$ by the secant method using $x = 1.2$ and 1.1 as starting values.
42. Solve Problem 41 by the secant method with $x = 0.5$ and 0.6 as starting values. You may want to solve this problem on a computer, since a large number of iterations may be required.
43. Solve Problems 35(a) to (c) by the secant method. Use the starting values given there for x_0 and let $x_1 = 1.1x_0$.
44. Solve Problem 36 using the secant method.

3.5. Polynomials

45. Use Newton's method to find the real roots of the following polynomials:
 - (a) $x^3 - 5x^2 + 7x - 3 = 0$
 - (b) $x^4 - 9x^3 + 24x^2 - 36x + 80 = 0$
 - (c) $x^3 - 2x^2 - 2x + 1 = 0$
 - (d) $3x^3 + 4x^2 - 8x - 2 = 0$
46. Use Newton's method to find the complex roots of the following polynomials:
 - (a) $x^4 - 9x^3 + 24x^2 - 36x + 80 = 0$
 - (b) $x^3 + 2x^2 + x + 2 = 0$
 - (c) $x^5 - 15x^4 + 85x^3 - 226x^2 + 274x - 120 = 0$

3.7 Systems of Nonlinear Equations

Solve the following systems of nonlinear equations using Newton's method.

47. $(x - 1)^2 + (y - 2)^2 = 3$ and $x^2/4 + y^2/3 = 1$. Solve for all roots.
48. $y = \cosh(x)$ and $x^2 + y^2 = 2$. Solve for both roots.
49. $x^2 + y^2 = 2x + y$ and $x^2/4 + y^2 = 1$. Solve for all four roots.
50. $y^2(1 - x) = x^3$ and $x^2 + y^2 = 1$.
51. $x^3 + y^3 - 3xy = 0$ and $x^2 + y^2 = 1$.
52. $(x^2 + y^2)^2 = 2xy$ and $y = x^3$.
53. $(2x)^{2/3} + y^{2/3} = (9)^{1/3}$ and $x^2/4 + y^2 = 1$.

3.8. Programs

54. Implement the Newton method program presented in Section 3.8.1. Check out the program using the given data set.
55. Work any of Problems 29 to 36 using the Newton method program.
56. Implement the secant method program presented in Section 3.8.2. Check out the program using the given data set.
57. Work any of Problems 37 to 44 using the secant method program.
58. Implement the Newton method program presented in Section 3.8.3 for solving simultaneous equations. Check out the program using the given data set.
59. Work any of Problems 5, 6, or 47 to 53 using the Newton method program.
60. Write a computer program to solve Freudenstein's equation, Eq. (3.3), by the secant method. Calculate ϕ for $\alpha = 40$ deg to 90 deg in increments $\Delta\alpha = 10$ deg. For $\alpha = 40$ deg, let $\phi_0 = 25$ deg and $\phi_1 = 30$ deg. For subsequent values of α , let ϕ_0 be the solution value for the previous value of α , and $\phi_1 = \phi_0 + 1.0$. Continue the calculations until ϕ changes by less than 0.00001 deg. Design the program output as illustrated in Example 3.5.
61. Write a computer program to solve the van der Waal equation of state, Eq. (G) in Problem 69, by the secant method. Follow the procedure described in Problem 69 to initiate the calculations. Design the program output as illustrated in Example 3.5. For $P = 10,000$ kPa, calculate v corresponding to $T = 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500$, and 1600 K. Write the program so that all the cases can be calculated in one run by stacking input data decks.
62. Write a computer program to solve the Colebrook equation, Eq. (I) in Problem 70, by the secant method for the friction coefficient f for specified value of the roughness ratio ε/D and the Reynolds number, Re . Use the approximation proposed by Genereaux (1939), Eq. (J), and 90 percent of that value as the initial approximations. Solve Problem 70 using the program.
63. Write a computer program to solve the $M - \varepsilon$ equation, Eq. (K) in Problem 71, by Newton's method for specified values of γ and ε . (a) Solve Problem 71, using the program. (b) Construct a table of M versus ε for $1.0 \leq \varepsilon \leq 10$, for subsonic flow, in increments $\Delta\varepsilon = 0.1$. For $\varepsilon = 1.1$, let $M_0 = 0.8$. For subsequent values of ε , let M_0 be the previous solution value.
64. Write a computer program to solve the $M - \varepsilon$ equation, Eq. (K) in Problem 71, by the secant method for specified values of γ and ε . (a) Solve Problem 71 using the program. (b) Construct a table of M versus ε for $1.0 \leq \varepsilon \leq 10$, for

supersonic flow, in increments $\Delta\varepsilon = 0.1$. For $\varepsilon = 1.1$ let $M_0 = 1.2$ and $M_1 = 1.3$. For subsequent values of ε , let M_0 be the previous solution value and $M_1 = 1.1M_0$.

65. Write a computer program to solve Eq. (L) in Problem 73 by the secant method for specified values of γ , δ , and M_1 . (a) Solve Problem 73 using the program. (b) Construct a table of M versus δ for $\gamma = 1.4$ and $M_1 = 1.0$, for $0 < \delta \leq 40$ deg, in increments $\Delta\delta = 1.0$ deg. For $\delta = 1.0$ deg, let $M_0 = 1.06$ and $M_1 = 1.08$. For subsequent values of δ , let M_0 be the previous solution value and $M_1 = 1.01M_0$.

APPLIED PROBLEMS

Several applied problems from various disciplines are presented in this section. All of these problems can be solved by any of the methods presented in this chapter. An infinite variety of exercises can be constructed by changing the numerical values of the parameters of the problem, by changing the starting values, or both.

66. Consider the four-bar linkage problem presented in Section 3.1. Solve for any (or all) of the results presented in Table 3.1.
67. Consider the four-bar linkage problem presented in Section 3.1. Rearrange this problem to solve for the value of r_1 such that $\phi = 60$ deg when $\alpha = 75$ deg. Numerous variations of this problem can be obtained by specifying combinations of ϕ and α .
68. Solve the four-bar linkage problem for $\theta_4 = 210$ deg by solving the two scalar components of the vector loop equation, Eq. (3.2), by Newton's method. Let the initial guesses be $\theta_2 = 20$ deg and $\theta_3 = 0$ deg. Continue the calculations until θ_2 and θ_3 change by less than 0.00001 deg. Show all calculations for the first iteration. Summarize the first iteration and subsequent iterations in a table, as illustrated in Example 3.12.
69. The van der Waal equation of state for a vapor is

$$\left(P + \frac{a}{v^2}\right)(v - b) = RT \quad (\text{F})$$

where P is the pressure ($\text{Pa} = \text{N/m}^2$), v is the specific volume (m^3/kg), T is the temperature (K), R is the gas constant (J/kg-K), and a and b are empirical constants. Consider water vapor, for which $R = 461.495 \text{ J/kg-K}$, $a = 1703.28 \text{ Pa} \cdot (\text{m}^3/\text{kg})^2$, and $b = 0.00169099 \text{ (m}^3/\text{kg)}$. Equation (F) can be rearranged into the form

$$Pv^3 - (Pb + RT)v^2 + av - ab = 0 \quad (\text{G})$$

Calculate the specific volume v for $P = 10,000 \text{ kPa}$ and $T = 800 \text{ K}$. Use the ideal gas law, $Pv = RT$, to obtain the initial guess (or guesses). Present the results in the format illustrated in the examples.

70. When an incompressible fluid flows steadily through a round pipe, the pressure drop due to the effects of wall friction is given by the empirical formula:

$$\Delta P = -0.5f\rho V^2 \left(\frac{L}{D}\right) \quad (\text{H})$$

where ΔP is the pressure drop, ρ is the density, V is the velocity, L is the pipe length, D is the pipe diameter, and f is the D'Arcy friction coefficient. Several empirical formulas exist for the friction coefficient f as a function of the dimensionless Reynolds number, $Re = DV\rho/\mu$, where μ is the viscosity. For flow in the turbulent regime between completely smooth pipe surfaces and wholly rough pipe surfaces, Colebrook (1939) developed the following empirical equation for the friction coefficient f :

$$\frac{1}{f^{1/2}} = -2 \log_{10} \left(\frac{\varepsilon/D}{3.7} + \frac{2.51}{Re f^{1/2}} \right) \quad (I)$$

where ε is the pipe surface roughness. Develop a procedure to determine f for specified values of ε/D and Re . Use the approximation proposed by Genereaux (1939) to determine the initial approximation(s):

$$f = 0.16 Re^{-0.16} \quad (J)$$

71. Solve for f for a pipe having $\varepsilon/D = 0.001$ for $Re = 10^n$, for $n = 4, 5, 6$, and 7 .
71. Consider quasi-one-dimensional isentropic flow of a perfect gas through a variable-area channel. The relationship between the Mach number M and the flow area A , derived by Zucrow and Hoffman [1976, Eq. (4.29)], is given by

$$\varepsilon = \frac{A}{A^*} = \frac{1}{M} \left[\frac{2}{\gamma + 1} \left(1 + \frac{\gamma - 1}{2} M^2 \right) \right]^{(\gamma+1)/2(\gamma-1)} \quad (K)$$

- where A^* is the choking area (i.e., the area where $M = 1$) and γ is the specific heat ratio of the flowing gas. For each value of ε , two values of M exist, one less than unity (i.e., subsonic flow) and one greater than unity (i.e., supersonic flow). Calculate both values of M for $\varepsilon = 10.0$ and $\gamma = 1.4$ by Newton's method. For the subsonic root, let $M_0 = 0.2$. For the supersonic root, let $M_0 = 5.0$.
72. Solve Problem 71 by the secant method. For the subsonic root, let $M_0 = 0.4$ and $M_1 = 0.6$. For the supersonic root, let $M_0 = 3.0$ and $M_1 = 4.0$.
73. Consider isentropic supersonic flow around a sharp expansion corner. The relationship between the Mach number before the corner (i.e., M_1) and after the corner (i.e., M_2), derived by Zucrow and Hoffman [1976, Eq. (8.11)], is given by

$$\delta = b^{1/2} \left(\tan^{-1} \left(\frac{M_2^2 - 1}{b} \right)^{1/2} - \tan^{-1} \left(\frac{M_1^2 - 1}{b} \right)^{1/2} \right) \\ - ((\tan^{-1}((M_2^2 - 1)^{1/2}) - \tan^{-1}((M_1^2 - 1)^{1/2})) \quad (L)$$

where $b = (\gamma + 1)/(\gamma - 1)$ and γ is the specific heat ratio of the gas. Develop a procedure to solve for M_2 for specified values of γ , δ , and M_1 . For $\gamma = 1.4$, solve for M_2 for the following combinations of M_1 and δ : (a) 1.0 and 10.0 deg, (b) 1.0 and 20.0 deg, (c) 1.5 and 10.0 deg, and (d) 1.5 and 20.0 deg. Use $M_2^{(0)} = 2.0$ and $M_2^{(1)} = 1.5$.

4

Polynomial Approximation and Interpolation

- 4.1 Introduction
- 4.2 Properties of Polynomials
- 4.3 Direct Fit Polynomials
- 4.4 Lagrange Polynomials
- 4.5 Divided Difference Tables and Divided Difference Polynomials
- 4.6 Difference Tables and Difference Polynomials
- 4.7 Inverse Interpolation
- 4.8 Multivariate Approximation
- 4.9 Cubic Splines
- 4.10 Least Squares Approximation
- 4.11 Programs
- 4.12 Summary
 Problems

Examples

- 4.1 Polynomial evaluation
- 4.2 Direct fit polynomials
- 4.3 Lagrange polynomials
- 4.4 Neville's method
- 4.5 Divided difference table
- 4.6 Divided difference polynomial
- 4.7 Difference table
- 4.8 Newton forward-difference polynomial
- 4.9 Newton backward-difference polynomial
- 4.10 Inverse interpolation
- 4.11 Successive univariate quadratic interpolation
- 4.12 Direct multivariate linear interpolation
- 4.13 Cubic splines
- 4.14 Least squares straight line approximation
- 4.15 Least squares quadratic polynomial approximation
- 4.16 Least squares quadratic bivariate polynomial approximation

4.1 INTRODUCTION

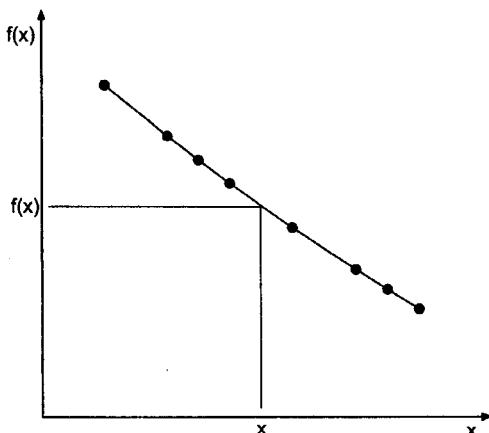
Figure 4.1 illustrates a set of tabular data in the form of a set of $[x, f(x)]$ pairs. The function $f(x)$ is known at a finite set (actually eight) of discrete values of x . The value of the function can be determined at any of the eight values of x simply by a table lookup. However, a problem arises when the value of the function is needed at any value of x between the discrete values in the table. The actual function is not known and cannot be determined from the tabular values. However, the actual function can be approximated by some known function, and the value of the approximating function can be determined at any desired value of x . This process, which is called *interpolation*, is the subject of Chapter 4. The discrete data in Figure 4.1 are actually values of the function $f(x) = 1/x$, which is used as the example problem in this chapter.

In many problems in engineering and science, the data being considered are known only at a set of discrete points, not as a continuous function. For example, the continuous function

$$y = f(x) \quad (4.1)$$

may be known only at n discrete values of x :

$$y_i = y(x_i) \quad (i = 1, 2, \dots, n) \quad (4.2)$$



x	$f(x)$
3.20	0.312500
3.30	0.303030
3.35	0.298507
3.40	0.294118
3.50	0.285714
3.60	0.277778
3.65	0.273973
3.70	0.270270

Figure 4.1 Approximation of tabular data.

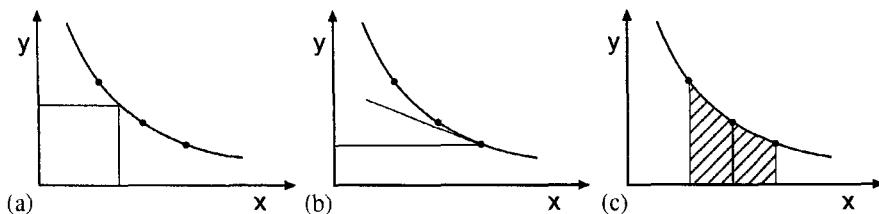


Figure 4.2 Applications of approximating functions. (a) Interpolation. (b) Differentiation. (c) Integration.

Discrete data, or tabular data, may consist of small sets of smooth data, large sets of smooth data, small sets of rough data, or large sets of rough data.

In many applications, the values of the discrete data at the specific points are not all that is needed. Values of the function at points other than the known discrete points may be needed (i.e., interpolation). The derivative of the function may be required (i.e., differentiation). The integral of the function may be of interest (i.e., integration). Thus, the processes of *interpolation*, *differentiation*, and *integration* of a set of discrete data are of interest. These processes are illustrated in Figure 4.2. These processes are performed by fitting an *approximating function* to the set of discrete data and performing the desired process on the approximating function.

Many types of approximating functions exist. In fact, any analytical function can be used as an approximating function. Three of the more common approximating functions are:

1. Polynomials
2. Trigonometric functions
3. Exponential functions

Approximating functions should have the following properties:

1. The approximating function should be easy to determine.
2. It should be easy to evaluate.
3. It should be easy to differentiate.
4. It should be easy to integrate.

Polynomials satisfy all four of these properties. Consequently, polynomial approximating functions are used in this book to fit sets of discrete data for interpolation, differentiation, and integration.

There are two fundamentally different ways to fit a polynomial to a set of discrete data:

1. Exact fits
2. Approximate fits

An *exact fit* yields a polynomial that passes exactly through all of the discrete points, as illustrated in Figure 4.3a. This type of fit is useful for small sets of smooth data. Exact polynomial fits are discussed in Sections 4.3 to 4.9. An *approximate fit* yields a polynomial that passes through the set of data in the best manner possible, without being required to pass exactly through any of the data points, as illustrated in Figure 4.3b. Several definitions of best manner possible exist. Approximate fits are useful for large sets of smooth data and

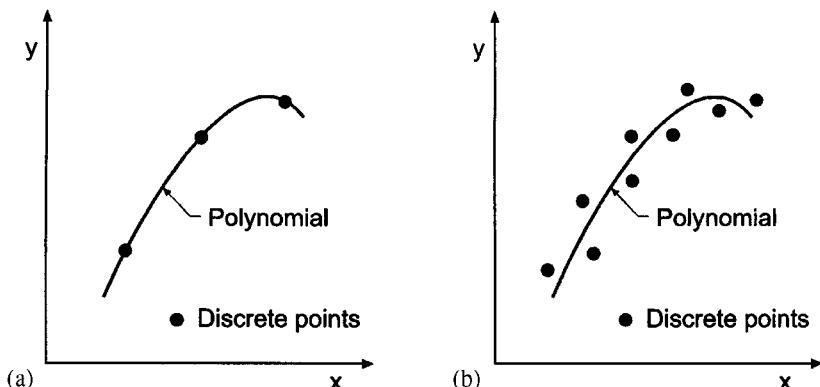


Figure 4.3 Polynomial approximation. (a) Exact fit. (b) Approximate fit.

small or large sets of rough data. In this book, the *least squares method* is used for approximate fits.

A set of discrete data may be *equally spaced* or *unequally spaced* in the independent variable x . In the general case where the data are unequally spaced, several procedures can be used to fit approximating polynomials, for example, (a) direct fit polynomials, (b) Lagrange polynomials, and (c) divided difference polynomials. Methods such as these require a considerable amount of effort. When the data are equally spaced, procedures based on differences can be used, for example, (a) the Newton forward-difference polynomial, (b) the Newton backward-difference polynomial, and (c) several other difference polynomials. These methods are quite easy to apply. Both types of methods are considered in this chapter.

Several procedures for polynomial approximation are developed in this chapter. Application of these procedures for interpolation is illustrated by examples. Numerical differentiation and numerical integration are discussed in Chapters 5 and 6, respectively.

Figure 4.4 illustrates the organization of Chapter 4. After the brief introduction in this section, the properties of polynomials which make them useful as approximating functions are presented. The presentation then splits into methods for fitting unequally spaced data and methods for fitting equally spaced data. A discussion of inverse interpolation follows next. Multivariate interpolation is then discussed. That is followed by an introduction to cubic splines. The final topic is a presentation of least squares approximation. Several programs for polynomial fitting are then presented. The chapter closes with a Summary which summarizes the main points of the chapter and presents a list of what you should be able to do after studying Chapter 4.

4.2 PROPERTIES OF POLYNOMIALS

The general form of an n th-degree polynomial is

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \quad (4.3)$$

where n denotes the degree of the polynomial and a_0 to a_n are constant coefficients. There are $n + 1$ coefficients, so $n + 1$ discrete data points are required to obtain unique values for the coefficients.

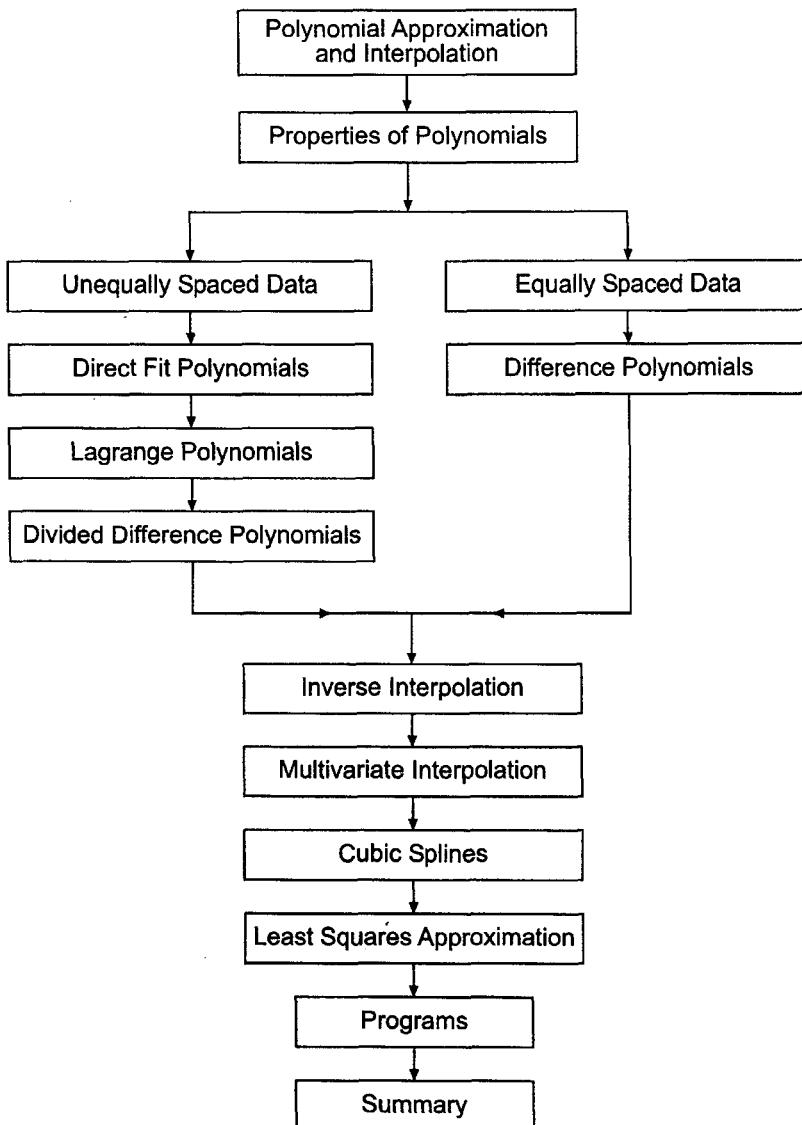


Figure 4.4 Organization of Chapter 4.

The property of polynomials that makes them suitable as approximating functions is stated by the *Weierstrass approximation theorem*:

If $f(x)$ is a continuous function in the closed interval $a \leq x \leq b$, then for every $\varepsilon > 0$ there exists a polynomial $P_n(x)$, where the value of n depends on the value of ε , such that for all x in the closed interval $a \leq x \leq b$,

$$|P_n(x) - f(x)| < \varepsilon$$

Consequently, any continuous function can be approximated to any accuracy by a polynomial of high enough degree. In practice, low-degree polynomials are employed, so care must be taken to achieve the desired accuracy.

Polynomials satisfy a *uniqueness theorem*:

A polynomial of degree n passing exactly through $n + 1$ discrete points is *unique*

The polynomial through a specific set of points may take many different forms, but all forms are equivalent. Any form can be manipulated into any other form by simple algebraic rearrangement.

The Taylor series is a polynomial of infinite order. Thus,

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!} f''(x_0)(x - x_0)^2 + \dots \quad (4.4)$$

It is, of course, impossible to evaluate an infinite number of terms. The Taylor polynomial of degree n is defined by

$$f(x) = P_n(x) + R_{n+1}(x) \quad (4.5)$$

where the Taylor polynomial $P_n(x)$, and the remainder term $R_{n+1}(x)$ are given by

$$P_n(x) = f(x_0) + f'(x_0)(x - x_0) + \dots + \frac{1}{n!} f^{(n)}(x_0)(x - x_0)^n \quad (4.6)$$

$$R_{n+1}(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x - x_0)^{n+1} \quad x_0 \leq \xi \leq x \quad (4.7)$$

The Taylor polynomial is a truncated Taylor series, with an explicit remainder, or error, term. The Taylor polynomial cannot be used as an approximating function for discrete data because the derivatives required in the coefficients cannot be determined. It does have great significance, however, for polynomial approximation, because it has an explicit error term.

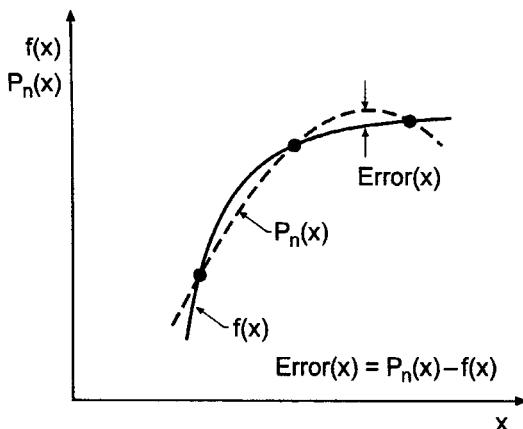
When a polynomial of degree n , $P_n(x)$, is fit exactly to a set of $n + 1$ discrete data points, $(x_0, f_0), (x_1, f_1), \dots, (x_n, f_n)$, as illustrated in Figure 4.5, the polynomial has no error at the data points themselves. However, at the locations between the data points, there is an error which is defined by

$$\text{Error}(x) = P_n(x) - f(x) \quad (4.8)$$

It can be shown that the *error term*, $\text{Error}(x)$, has the form

$$\text{Error}(x) = \frac{1}{(n+1)!} (x - x_0)(x - x_1) \cdots (x - x_n) f^{(n+1)}(\xi) \quad (4.9)$$

where $x_0 \leq \xi \leq x_n$. This form of the error term is used extensively in the error analysis of procedures based on approximating polynomials. Equation (4.9) shows that the error in any polynomial approximation of discrete data (e.g., interpolation, differentiation, or integration) will be the smallest possible when the approximation is centered in the discrete data because that makes the $(x - x_i)$ terms as small as possible, which makes the product of those terms the smallest possible.

**Figure 4.5** Error in polynomial approximation.

Differentiation of polynomials is straightforward. For the general term $a_i x^i$,

$$\frac{d}{dx}(a_i x^i) = i a_i x^{i-1} \quad (4.10)$$

The derivatives of the n th-degree polynomial $P_n(x)$ are

$$\frac{dP_n(x)}{dx} = P'_n(x) = a_1 + 2a_2 x + \cdots + n a_n x^{n-1} = P_{n-1}(x) \quad (4.11a)$$

$$\frac{d^2 P_n(x)}{dx^2} = \frac{d}{dx} \left[\frac{dP_n(x)}{dx} \right] = P''_n(x) = 2a_2 + 6a_3 x + \cdots + n(n-1)a_n x^{n-2} = P_{n-2}(x) \quad (4.11b)$$

.....

$$P_n^{(n)}(x) = n! a_n \quad (4.11n)$$

$$P_n^{(n+1)}(x) = 0 \quad (4.12)$$

Integration of polynomials is equally straightforward. For the general term $a_i x^i$,

$$\int a_i x^i dx = \frac{a_i}{i+1} x^{i+1} + \text{constant} \quad (4.13)$$

The integral of the n th-degree polynomial $P_n(x)$ is

$$I = \int P_n(x) dx = \int (a_0 + a_1 x + \cdots + a_n x^n) dx \quad (4.14)$$

$$I = a_0 x + \frac{a_1}{2} x^2 + \cdots + \frac{a_n}{n+1} x^{n+1} + \text{constant} = P_{n+1}(x) \quad (4.15)$$

The *evaluation of a polynomial*, $P_n(x)$, its derivative, $P'_n(x)$, or its integral, $\int P_n(x) dx$, for a particular value of x , is straightforward. For example, consider the fourth-degree polynomial, $P_4(x)$, its derivative, $P'_4(x)$, and its integral, $\int P_4(x) dx$:

$$P_4(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 \quad (4.16a)$$

$$P'_4(x) = a_1 + 2a_2x + 3a_3x^2 + 4a_4x^3 = P_3(x) \quad (4.16b)$$

$$\int P_4(x) dx = a_0x + \frac{a_1}{2}x^2 + \frac{a_2}{3}x^3 + \frac{a_3}{4}x^4 + \frac{a_4}{5}x^5 + \text{constant} = P_5(x) \quad (4.16c)$$

The evaluation of Eq. (4.16a) requires $(0 + 1 + 2 + 3 + 4) = 10$ multiplications and four additions; the evaluation of Eq. (4.16b) requires $(2 + 3 + 4) = 9$ multiplications and three additions; and the evaluation of Eq. (4.16c) requires $(1 + 2 + 3 + 4 + 5) = 15$ multiplications, four divisions, and five additions. This is a modest amount of work, even for polynomials of degree as high as 10. However, if a polynomial must be evaluated many times, or many polynomials must be evaluated, or very high degree polynomials must be evaluated, a more efficient procedure is desirable. The nested multiplication algorithm is such a procedure.

The *nested multiplication algorithm* is based on the following rearrangement of Eq. (4.16a):

$$P_4(x) = a_0 + x[a_1 + x[a_2 + x(a_3 + a_4x)]] \quad (4.17)$$

which requires four multiplications and four additions. For a polynomial of degree n , $P_n(x)$, *nested multiplication* is given by

$$P_n(x) = a_0 + x(a_1 + x(a_2 + x[a_3 + \cdots + x(a_{n-1} + a_nx)])) \quad (4.18)$$

which requires n multiplications and n additions. Equation (4.18) can be evaluated by constructing the *nested multiplication* sequence:

$$\boxed{\begin{aligned} b_n &= a_n \\ b_i &= a_i + xb_{i+1} \quad (i = n-1, n-2, \dots, 0) \end{aligned}} \quad (4.19)$$

where $P_n(x) = b_0$. Equations (4.16b) and (4.16c) can be evaluated in a similar manner with minor modifications to account for the proper coefficients. Nested multiplication is sometimes called *Horner's algorithm*.

Several other properties of polynomials are quite useful. The *division algorithm* states that

$$P_n(x) = (x - N)Q_{n-1}(x) + R \quad (4.20)$$

where N is any number, $Q_{n-1}(x)$ is a polynomial of degree $n - 1$, and R is a constant remainder. The *remainder theorem* states that

$$P_n(N) = R \quad (4.21)$$

The *factor theorem* states that if $P_n(N) = 0$, then $(x - N)$ is a factor of $P_n(x)$, which means that N is a root, α , or zero, of $P_n(x)$. That is, $(x - N) = 0$, and $\alpha = N$.

The *derivative of a polynomial* $P'_n(x)$ can be obtained from Eq. (4.20). Thus,

$$P'_n(x) = Q_{n-1}(x) + (x - N)Q'_{n-1}(x) \quad (4.22)$$

At $x = N$,

$$P'_n(N) = Q_{n-1}(N) \quad (4.23)$$

Consequently, first derivatives of an n th-degree polynomial can be evaluated from the $(n - 1)$ st-degree polynomial $Q_{n-1}(x)$. Higher-order derivatives can be determined by applying the synthetic division algorithm to $Q_{n-1}(x)$, etc.

The $(n - 1)$ st-degree polynomial $Q_{n-1}(x)$, which can be used to evaluate the derivative $P'_n(x)$ and the remainder R , which yields $P_n(N) = R$, can be evaluated by the *synthetic division algorithm*. Consider $P_n(x)$ in the form given by Eq. (4.3):

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \quad (4.24a)$$

and $Q_{n-1}(x)$ in the form

$$Q_{n-1}(x) = b_1 + b_2x + b_3x^2 + \cdots + b_{n-1}x^{n-2} + b_nx^{n-1} \quad (4.24b)$$

Substituting Eqs. (4.24a) and (4.24b) into Eq. (4.20) and equating coefficients of like powers of x yields:

$$b_n = a_n \quad (4.25.n)$$

$$b_{n-1} = a_{n-1} + xb_n \quad (4.25.n-1)$$

.....

$$b_1 = a_1 + xb_2 \quad (4.25.1)$$

$$b_0 = a_0 + xb_1 = R \quad (4.25.0)$$

Equation (4.25) can be written as

$$\boxed{b_n = a_n \\ b_i = a_i + xb_{i+1} \quad (i = n - 1, n - 2, \dots, 0)} \quad (4.26)$$

Equation (4.26) is identical to the nested multiplication algorithm presented in Eq. (4.19). Substituting $x = N$ into Eq. (4.24b) yields the value of $P'_n(N)$.

If a root, α , or zero, of $P_n(x)$ is known, $P_n(x)$ can be *deflated* by removing the factor $(x - \alpha)$ to yield the $(n - 1)$ st-degree polynomial, $Q_{n-1}(x)$. From Eq. (4.20), if α is a root of $P_n(x)$, then $P_n(\alpha) = 0$ and $R = 0$, and Eq. (4.20) yields

$$Q_{n-1}(x) = 0 \quad (4.27)$$

The deflated polynomial $Q_{n-1}(x)$ has $n - 1$ roots, or zeros, which are the remaining roots, or zeros, of the original polynomial, $P_n(x)$.

The properties of polynomials presented in this section make them extremely useful as approximating functions.

Example 4.1. Polynomial evaluation.

Let's illustrate polynomial evaluation using nested multiplication, polynomial derivative evaluation using synthetic division, and polynomial deflation using synthetic division. Consider the fifth-degree polynomial considered in Section 3.5.1, Eq. (3.115):

$$P_5(x) = -120 + 274x - 225x^2 + 85x^3 - 15x^4 + x^5 \quad (4.28)$$

Recall that the roots of Eq. (4.28) are $x = 1, 2, 3, 4$, and 5 . Evaluate $P_5(2.5)$ and $P'_5(2.5)$, and determine the deflated polynomial $P_4(x)$ obtained by removing the factor $(x - 2)$.

Evaluating $P_5(2.5)$ by nested multiplication using Eq. (4.19) yields

$$b_5 = 1.0 \quad (4.29.5)$$

$$b_4 = -15.0 + 2.5(1.0) = -12.50 \quad (4.29.4)$$

$$b_3 = 85.0 + 2.5(-12.5) = 53.750 \quad (4.29.3)$$

$$b_2 = -225.0 + 2.5(53.750) = -90.6250 \quad (4.29.2)$$

$$b_1 = 274.0 + 2.5(-90.6250) = 47.43750 \quad (4.29.1)$$

$$b_0 = -120.0 + 2.5(47.43750) = -1.406250 \quad (4.29.0)$$

Thus, $P_5(2.50) = b_0 = -1.406250$. This result can be verified by direct evaluation of Eq. (4.28) with $x = 2.5$.

From Eq. (4.23), $P'_5(2.5) = Q_4(2.5)$, where $Q_4(x)$ is given by

$$Q_4(x) = 47.43750 - 90.6250x + 53.750x^2 - 12.50x^3 + x^4 \quad (4.30)$$

Evaluating $Q_4(2.5)$ by nested multiplication using Eq. (4.19), with the b_i replaced by c_i , gives

$$c_4 = 1.0 \quad (4.31.4)$$

$$c_3 = -12.5 + 2.5(1.0) = -10.0 \quad (4.31.3)$$

$$c_2 = 53.75 + 2.5(-10.0) = 28.750 \quad (4.31.2)$$

$$c_1 = -90.625 + 2.5(28.750) = -18.750 \quad (4.31.1)$$

$$c_0 = 47.4375 + 2.5(-18.750) = 0.56250 \quad (4.31.0)$$

Thus, $P'_5(2.5) = Q_4(2.5) = c_0 = 0.56250$. This result can be verified by direct evaluation of Eq. (4.30) with $x = 2.5$.

To illustrate polynomial deflation, let's deflate $P_5(x)$ by removing the factor $(x - 2)$. Applying the synthetic division algorithm, Eq. (4.26), with $x = 2.0$ yields

$$b_5 = 1.0 \quad (4.32.5)$$

$$b_4 = -15.0 + 2.0(1.0) = -13.0 \quad (4.32.4)$$

$$b_3 = 85.0 + 2.0(-13.0) = 59.0 \quad (4.32.3)$$

$$b_2 = -225.0 + 2.0(59.0) = -107.0 \quad (4.32.2)$$

$$b_1 = 274.0 + 2.0(-107.0) = 60.0 \quad (4.32.1)$$

$$b_0 = -120.0 + 2.0(60.0) = 0.0 \quad (4.32.0)$$

Thus, the deflated fourth-degree polynomial is

$$Q_4(x) = 60.0 - 107.0x + 59.0x^2 - 13.0x^3 + x^4 \quad (4.33)$$

This result can be verified directly by expanding the product of the four remaining linear factors, $Q_4(x) = (x - 1)(x - 3)(x - 4)(x - 5)$.

4.3 DIRECT FIT POLYNOMIALS

First let's consider a completely general procedure for fitting a polynomial to a set of equally spaced or unequally spaced data. Given $n + 1$ sets of data $[x_0, f(x_0)]$, $[x_1, f(x_1)]$, \dots , $[x_n, f(x_n)]$, which will be written as (x_0, f_0) , (x_1, f_1) , \dots , (x_n, f_n) , determine the unique n th-degree polynomial $P_n(x)$ that passes exactly through the $n + 1$ points:

$$P_n(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n \quad (4.34)$$

For simplicity of notation, let $f(x_i) = f_i$. Substituting each data point into Eq. (4.34) yields $n + 1$ equations:

$$f_0 = a_0 + a_1x_0 + a_2x_0^2 + \dots + a_nx_0^n \quad (4.35.0)$$

$$f_1 = a_0 + a_1x_1 + a_2x_1^2 + \dots + a_nx_1^n \quad (4.35.1)$$

.....

$$f_n = a_0 + a_1x_n + a_2x_n^2 + \dots + a_nx_n^n \quad (4.35.n)$$

There are $n + 1$ linear equations containing the $n + 1$ coefficients a_0 to a_n . Equation (4.35) can be solved for a_0 to a_n by Gauss elimination. The resulting polynomial is the unique n th-degree polynomial that passes exactly through the $n + 1$ data points. The direct fit polynomial procedure works for both equally spaced data and unequally spaced data.

Example 4.2. Direct fit polynomials.

To illustrate interpolation by a direct fit polynomial, consider the simple function $y = f(x) = 1/x$, and construct the following set of six significant figure data:

x	$f(x)$
3.35	0.298507
3.40	0.294118
3.50	0.285714
3.60	0.277778

Let's interpolate for y at $x = 3.44$ using linear, quadratic, and cubic interpolation. The exact value is

$$y(3.44) = f(3.44) = \frac{1}{3.44} = 0.290698\dots \quad (4.36)$$

Let's illustrate the procedure in detail for a quadratic polynomial:

$$P_2(x) = a + bx + cx^2 \quad (4.37)$$

To center the data around $x = 3.44$, the first three points are used. Applying $P_2(x)$ at each of these data points gives the following three equations:

$$0.298507 = a + b(3.35) + c(3.35)^2 \quad (4.38.1)$$

$$0.294118 = a + b(3.40) + c(3.40)^2 \quad (4.38.2)$$

$$0.285714 = a + b(3.50) + c(3.50)^2 \quad (4.38.3)$$

Solving Eqs. (4.38) for a , b , and c by Gauss elimination without scaling or pivoting yields

$$P_2(x) = 0.876561 - 0.256080x + 0.0249333x^2 \quad (4.39)$$

Substituting $x = 3.44$ into Eq. (4.39) gives

$$P_2(3.44) = 0.876561 - 0.256080(3.44) + 0.0249333(3.44)^2 = 0.290697 \quad (4.40)$$

The error is Error(3.44) = $P_2(3.44) - f(3.44) = 0.290697 - 0.290698 = -0.000001$.

For a linear polynomial, use $x = 3.40$ and 3.50 to center that data around $x = 3.44$. The resulting linear polynomial is

$$P_1(x) = 0.579854 - 0.0840400x \quad (4.41)$$

Substituting $x = 3.44$ into Eq. (4.41) gives $P_1(3.44) = 0.290756$. For a cubic polynomial, all four points must be used. The resulting cubic polynomial is

$$P_3(x) = 1.121066 - 0.470839x + 0.0878000x^2 - 0.00613333x^3 \quad (4.42)$$

Substituting $x = 3.44$ into Eq. (4.42) gives $P_3(3.44) = 0.290698$.

The results are summarized below, where the results of linear, quadratic, and cubic interpolation, and the errors, Error(3.44) = $P(3.44) - 0.290698$, are tabulated. The advantages of higher-degree interpolation are obvious.

$P(3.44) = 0.290756$	linear interpolation	Error = 0.000058
$= 0.290697$	quadratic interpolation	= -0.000001
$= 0.290698$	cubic interpolation	= 0.000000

The main advantage of direct fit polynomials is that the explicit form of the approximating function is obtained, and interpolation at several values of x can be accomplished simply by evaluating the polynomial at each value of x . The work required to obtain the polynomial does not have to be redone for each value of x . A second advantage is that the data can be unequally spaced.

The main disadvantage of direct fit polynomials is that each time the degree of the polynomial is changed, all of the work required to fit the new polynomial must be redone. The results obtained from fitting other degree polynomials is of no help in fitting the next polynomial. One approach for deciding when polynomial interpolation is accurate enough is to interpolate with successively higher-degree polynomials until the change in the result is within an acceptable range. This procedure is quite laborious using direct fit polynomials.

4.4 LAGRANGE POLYNOMIALS

The direct fit polynomial presented in Section 4.3, while quite straightforward in principle, has several disadvantages. It requires a considerable amount of effort to solve the system

of equations for the coefficients. For a high-degree polynomial (n greater than about 4), the system of equations can be ill-conditioned, which causes large errors in the values of the coefficients. A simpler, more direct procedure is desired. One such procedure is the *Lagrange polynomial*, which can be fit to unequally spaced data or equally spaced data. The Lagrange polynomial is presented in Section 4.4.1. A variation of the Lagrange polynomial, called *Neville's algorithm*, which has some computational advantages over the Lagrange polynomial, is presented in Section 4.4.2.

4.4.1. Lagrange Polynomials

Consider two points, $[a, f(a)]$ and $[b, f(b)]$. The linear Lagrange polynomial $P_1(x)$ which passes through these two points is given by

$$P_1(x) = \frac{(x - b)}{(a - b)}f(a) + \frac{(x - a)}{(b - a)}f(b) \quad (4.43)$$

Substituting $x = a$ and $x = b$ into Eq. (4.43) yields

$$P_1(a) = \frac{(a - b)}{(a - b)}f(a) + \frac{(a - a)}{(b - a)}f(b) = f(a) \quad (4.44a)$$

$$P_1(b) = \frac{(b - b)}{(a - b)}f(a) + \frac{(b - a)}{(b - a)}f(b) = f(b) \quad (4.44b)$$

which demonstrates that Eq. (4.43) passes through the two points. Given three points, $[a, f(a)]$, $[b, f(b)]$, and $[c, f(c)]$, the quadratic Lagrange polynomial $P_2(x)$ which passes through the three points is given by:

$$P_2(x) = \frac{(x - b)(x - c)}{(a - b)(a - c)}f(a) + \frac{(x - a)(x - c)}{(b - a)(b - c)}f(b) + \frac{(x - a)(x - b)}{(c - a)(c - b)}f(c) \quad (4.45)$$

Substitution of the values of x substantiates that Eq. (4.45) passes through the three points.

This procedure can be applied to any set of $n + 1$ points to determine an n th-degree polynomial. Given $n + 1$ points, $[a, f(a)]$, $[b, f(b)]$, ..., $[k, f(k)]$, the n th degree Lagrange polynomial $P_n(x)$ which passes through the $n + 1$ points is given by:

$$P_n(x) = \frac{(x - b)(x - c) \cdots (x - k)}{(a - b)(a - c) \cdots (a - k)}f(a) + \frac{(x - a)(x - c) \cdots (x - k)}{(b - a)(b - c) \cdots (b - k)}f(b) \\ + \cdots + \frac{(x - a)(x - b) \cdots (x - j)}{(k - a)(k - b) \cdots (k - j)}f(k) \quad (4.46)$$

The Lagrange polynomial can be used for both unequally spaced data and equally spaced data. No system of equations must be solved to evaluate the polynomial. However, a considerable amount of computational effort is involved, especially for higher-degree polynomials.

The form of the Lagrange polynomial is quite different in appearance from the form of the direct fit polynomial, Eq. (4.34). However, by the uniqueness theorem, the two forms both represent the unique polynomial that passes exactly through a set of points.

Example. 4.3. Lagrange polynomials.

Consider the four points given in Example 4.2, which satisfy the simple function $y = f(x) = 1/x$:

x	$f(x)$
3.35	0.298507
3.40	0.294118
3.50	0.285714
3.60	0.277778

Let's interpolate for $y = f(3.44)$ using linear, quadratic, and cubic Lagrange interpolating polynomials. The exact value is $y = 1/3.44 = 0.290698 \dots$

Linear interpolation using the two closest points, $x = 3.40$ and 3.50 , yields

$$P_1(3.44) = \frac{(3.44 - 3.50)}{(3.40 - 3.50)}(0.294118) + \frac{(3.44 - 3.40)}{(3.50 - 3.40)}(0.285714) = 0.290756 \quad (4.47)$$

Quadratic interpolation using the three closest points, $x = 3.35$, 3.40 , and 3.50 , gives

$$\begin{aligned} P_2(3.44) &= \frac{(3.44 - 3.40)(3.44 - 3.50)}{(3.35 - 3.40)(3.35 - 3.50)}(0.298507) \\ &\quad + \frac{(3.44 - 3.35)(3.44 - 3.50)}{(3.40 - 3.35)(3.40 - 3.50)}(0.294118) \\ &\quad + \frac{(3.44 - 3.35)(3.44 - 3.40)}{(3.50 - 3.35)(3.50 - 3.40)}(0.285714) = 0.290697 \end{aligned} \quad (4.48)$$

Cubic interpolation using all four points yields

$$\begin{aligned} P_3(3.44) &= \frac{(3.44 - 3.40)(3.44 - 3.50)(3.44 - 3.60)}{(3.35 - 3.40)(3.35 - 3.50)(3.35 - 3.60)}(0.298507) \\ &\quad + \frac{(3.44 - 3.35)(3.44 - 3.50)(3.44 - 3.60)}{(3.40 - 3.35)(3.40 - 3.50)(3.40 - 3.60)}(0.294118) \\ &\quad + \frac{(3.44 - 3.35)(3.44 - 3.40)(3.44 - 3.60)}{(3.50 - 3.35)(3.50 - 3.40)(3.50 - 3.60)}(0.285714) \\ &\quad + \frac{(3.44 - 3.35)(3.44 - 3.40)(3.44 - 3.50)}{(3.60 - 3.35)(3.60 - 3.40)(3.60 - 3.50)}(0.277778) = 0.290698 \end{aligned} \quad (4.49)$$

The results are summarized below, where the results of linear, quadratic, and cubic interpolation, and the errors, $\text{Error}(3.44) = P(3.44) - 0.290698$, are tabulated. The advantages of higher-degree interpolation are obvious.

$P(3.44) = 0.290756$	$\text{linear interpolation}$	$\text{Error} = 0.000058$
$= 0.290697$	$\text{quadratic interpolation}$	$= -0.000001$
$= 0.290698$	$\text{cubic interpolation}$	$= 0.000000$

These results are identical to the results obtained in Example 4.2 by direct fit polynomials, as they should be, since the same data points are used in both examples.

The main advantage of the Lagrange polynomial is that the data may be unequally spaced. There are several disadvantages. All of the work must be redone for each degree polynomial. All the work must be redone for each value of x . The first disadvantage is eliminated by Neville's algorithm, which is presented in the next subsection. Both disadvantages are eliminated by using divided differences, which are presented in Section 4.5.

4.4.2. Neville's Algorithm

Neville's algorithm is equivalent to a Lagrange polynomial. It is based on a series of linear interpolations. The data do not have to be in monotonic order, or in any structured order. However, the most accurate results are obtained if the data are arranged in order of closeness to the point to be interpolated.

Consider the following set of data:

x_i	f_i
x_1	f_1
x_2	f_2
x_3	f_3
x_4	f_4

Recall the linear Lagrange interpolating polynomial, Eq. (4.43):

$$f(x) = \frac{(x - b)}{(a - b)} f(a) + \frac{(x - a)}{(b - a)} f(b) \quad (4.50)$$

which can be written in the following form:

$$f(x) = \frac{(x - a)f(b) - (x - b)f(a)}{(b - a)} \quad (4.51)$$

In terms of general notation, Eq. (4.51) yields

$$f_i^{(n)} = \frac{(x - x_i)f_{i+1}^{(n-1)} - (x - x_{i+n})f_i^{(n-1)}}{x_{i+n} - x_i} \quad (4.52)$$

where the subscript i denotes the base point of the value (e.g., $i, i+1$, etc.) and the superscript (n) denotes the degree of the interpolation (e.g., zeroth, first, second, etc.).

A table of linearly interpolated values is constructed for the original data, which are denoted as $f_i^{(0)}$. For the first interpolation of the data,

$$f_i^{(1)} = \frac{(x - x_i)f_{i+1}^{(0)} - (x - x_{i+1})f_i^{(0)}}{x_{i+1} - x_i} \quad (4.53)$$

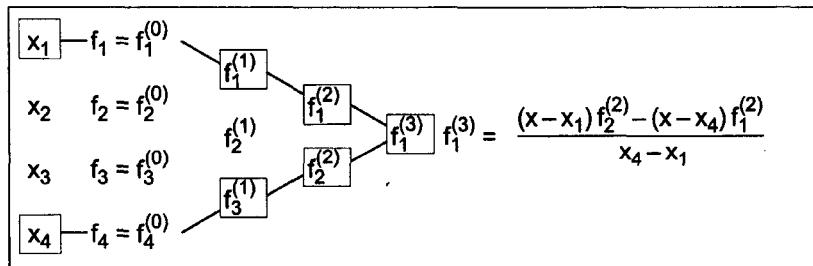
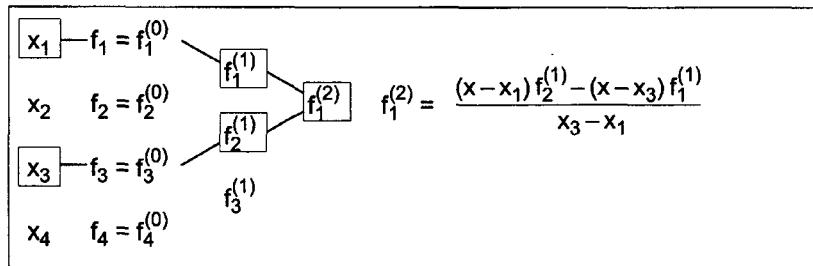
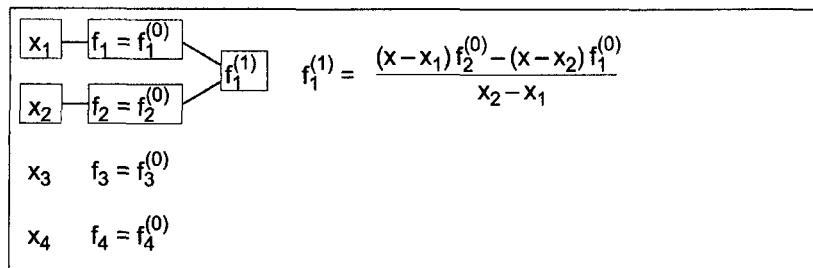


Figure 4.6 Neville's method. (a) First set of linear interpolations. (b) Second set of linear interpolation. (c) Third set of linear interpolations

as illustrated in Figure 4.6a. This creates a column of $n - 1$ values of $f_i^{(1)}$. A second column of $n - 2$ values of $f_i^{(2)}$ is obtained by linearly interpolating the column of $f_i^{(1)}$ values. Thus,

$$f_i^{(2)} = \frac{(x - x_i)f_{i+1}^{(1)} - (x - x_{i+2})f_i^{(1)}}{x_{i+2} - x_i} \quad (4.54)$$

which is illustrated in Figure 4.6b. This process is repeated to create a third column of $f_i^{(3)}$ values, as illustrated in Figure 4.6c, and so on. The form of the resulting table is illustrated in Table 4.1.

It can be shown by direct substitution that each specific value in Table 4.1 is identical to a Lagrange polynomial based on the data points used to calculate the specific value. For example, $f_1^{(2)}$ is identical to a second-degree Lagrange polynomial based on points 1, 2, and 3.

The advantage of Neville's algorithm over direct Lagrange polynomial interpolation is now apparent. The third-degree Lagrange polynomial based on points 1 to 4 is obtained simply by applying the linear interpolation formula, Eq. (4.52), to $f_1^{(2)}$ and $f_2^{(2)}$ to obtain

Table 4.1. Table for Neville's Algorithm

x_i	$f_i^{(0)}$	$f_i^{(1)}$	$f_i^{(2)}$	$f_i^{(3)}$
x_1	$f_1^{(0)}$			
x_2	$f_2^{(0)}$	$f_1^{(1)}$	$f_1^{(2)}$	
x_3	$f_3^{(0)}$	$f_2^{(1)}$	$f_2^{(2)}$	$f_1^{(3)}$
x_4	$f_4^{(0)}$	$f_3^{(1)}$		

$f_1^{(3)}$. None of the prior work must be redone, as it would have to be redone to evaluate a third-degree Lagrange polynomial. If the original data are arranged in order of closeness to the interpolation point, each value in the table, $f_i^{(n)}$, represents a centered interpolation.

Example 4.4. Neville's algorithm.

Consider the four data points given in Example 4.3. Let's interpolate for $f(3.44)$ using linear, quadratic, and cubic interpolation using Neville's algorithm. Rearranging the data in order of closeness to $x = 3.44$ yields the following set of data:

x	$f(x)$
3.40	0.294118
3.50	0.285714
3.35	0.298507
3.60	0.277778

Applying Eq. (4.52) to the values of $f_i^{(0)}$ gives

$$f_1^{(1)} = \frac{(x - x_1)f_2^{(0)} - (x - x_2)f_1^{(0)}}{x_2 - x_1} = \frac{(3.44 - 3.40)0.285714 - (3.44 - 3.50)0.294118}{3.50 - 3.40} \\ = 0.290756 \quad (4.55a)$$

Thus, the result of linear interpolation is $f(3.44) = f_1^{(1)} = 0.290756$. To evaluate $f_1^{(2)}, f_2^{(1)}$ must first be evaluated. Thus,

$$f_2^{(1)} = \frac{(x - x_2)f_3^{(0)} - (x - x_3)f_2^{(0)}}{x_3 - x_2} = \frac{(3.44 - 3.50)0.298507 - (3.44 - 3.35)0.285714}{3.35 - 3.50} \\ = 0.290831 \quad (4.55b)$$

Evaluating $f_1^{(2)}$ gives

$$f_1^{(2)} = \frac{(x - x_1)f_2^{(1)} - (x - x_3)f_1^{(1)}}{x_3 - x_1} = \frac{(3.44 - 3.40)0.290831 - (3.44 - 3.35)0.290756}{3.35 - 3.40} \\ = 0.290696 \quad (4.56)$$

Table 4.2. Neville's Algorithm

x_i	$f_i^{(0)}$	$f_i^{(1)}$	$f_i^{(2)}$	$f_i^{(3)}$
$x_1 = 3.40$	0.294118			
$x_2 = 3.50$	0.285714	0.290756	0.290697	
$x_3 = 3.35$	0.298507	0.290831	0.290703	0.290698
$x_4 = 3.60$	0.277778	0.291045		

Thus, the result of quadratic interpolation is $f(3.44) = f_1^{(2)} = 0.290696$. To evaluate $f_1^{(3)}, f_3^{(1)}$ and $f_2^{(2)}$ must first be evaluated. Then $f_1^{(3)}$ can be evaluated. These results, and the results calculated above, are presented in Table 4.2.

These results are the same as the results obtained by Lagrange polynomials in Example 4.3.

The advantage of Neville's algorithm over a Lagrange interpolating polynomial, if the data are arranged in order of closeness to the interpolated point, is that none of the work performed to obtain a specific degree result must be redone to evaluate the next higher degree result.

Neville's algorithm has a couple of minor disadvantages. All of the work must be redone for each new value of x . The amount of work is essentially the same as for a Lagrange polynomial. The divided difference polynomial presented in Section 4.5 minimizes these disadvantages.

4.5 DIVIDED DIFFERENCE TABLES AND DIVIDED DIFFERENCE POLYNOMIALS

A *divided difference* is defined as the ratio of the difference in the function values at two points divided by the difference in the values of the corresponding independent variable. Thus, the first divided difference at point i is defined as

$$f[x_i, x_{i+1}] = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} \quad (4.57)$$

The second divided difference is defined as

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i} \quad (4.58)$$

Similar expressions can be obtained for divided differences of any order. Approximating polynomials for nonequally spaced data can be constructed using divided differences.

4.5.1. Divided Difference Tables

Consider a table of data:

x_i	f_i
x_0	f_0
x_1	f_1
x_2	f_2
x_3	f_3

The first divided differences, in terms of standard notation, are

$$f[x_0, x_1] = \frac{(f_1 - f_0)}{(x_1 - x_0)} \quad (4.59a)$$

$$f[x_1, x_2] = \frac{(f_2 - f_1)}{(x_2 - x_1)} \quad (4.59b)$$

etc. Note that

$$f[x_i, x_{i+1}] = \frac{(f_{i+1} - f_i)}{(x_{i+1} - x_i)} = \frac{(f_i - f_{i+1})}{(x_i - x_{i+1})} = f[x_{i+1}, x_i] \quad (4.60)$$

The second divided difference is defined as follows:

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{(x_2 - x_0)} \quad (4.61)$$

In general,

$$f[x_0, x_1, \dots, x_n] = \frac{f[x_1, x_2, \dots, x_n] - f[x_0, x_1, \dots, x_{n-1}]}{(x_n - x_0)} \quad (4.62)$$

By definition, $f[x_i] = f_i$.

The notation presented above is a bit clumsy. A more compact notation is defined in the same manner as the notation used in Neville's method, which is presented in Section 4.4.2. Thus,

$$f_i^{(1)} = f[x_i, x_{i+1}] \quad (4.63a)$$

$$f_i^{(2)} = f[x_i, x_{i+1}, x_{i+2}] \quad (4.63b)$$

In general,

$$f_i^{(n)} = f[x_i, x_{i+1}, \dots, x_{i+n}] \quad (4.64)$$

Table 4.3 illustrates the formation of a divided difference table. The first column contains the values of x_i and the second column contains the values of $f(x_i) = f_i$, which are denoted by $f_i^{(0)}$. The remaining columns contain the values of the divided differences, $f_i^{(n)}$, where the subscript i denotes the base point of the value and the superscript (n) denotes the degree of the divided difference.

The data points do not have to be in any specific order to apply the divided difference concept. However, just as for the direct fit polynomial, the Lagrange polynomial, and Neville's method, more accurate results are obtained if the data are arranged in order of closeness to the interpolated point.

Table 4.3. Table of Divided differences

x_i	$f_i^{(0)}$	$f_i^{(1)}$	$f_i^{(2)}$	$f_i^{(3)}$
x_1	$f_1^{(0)}$			
x_2	$f_2^{(0)}$	$f_1^{(1)}$	$f_1^{(2)}$	
x_3	$f_3^{(0)}$	$f_2^{(1)}$	$f_2^{(2)}$	$f_1^{(3)}$
x_4	$f_4^{(0)}$	$f_3^{(1)}$		

Example 4.5. Divided difference Table.

Let's construct a six-place divided difference table for the data presented in Section 4.1. The results are presented in Table 4.4.

Table 4.4. Divided Difference Table

x_i	$f_i^{(0)}$	$f_i^{(1)}$	$f_i^{(2)}$	$f_i^{(3)}$	$f_i^{(4)}$
3.20	0.312500				
3.30	0.303030	-0.094700	0.028267		
3.35	0.298507	-0.090460	0.026800	-0.007335	-0.000667
3.40	0.294118	-0.087780	0.024933	-0.009335	0.010677
3.50	0.285714	-0.084040	0.023400	-0.006132	-0.001787
3.60	0.277778	-0.079360	0.021733	-0.006668	0.000010
3.65	0.273973	-0.076100	0.020400	-0.006665	
3.70	0.270270	-0.074060			

4.5.2. Divided Difference Polynomials

Let's define a power series for $P_n(x)$ such that the coefficients are identical to the divided differences, $f_i^{(n)}$. Thus,

$$\boxed{P_n(x) = f_i^{(0)} + (x - x_0)f_i^{(1)} + (x - x_0)(x - x_1)f_i^{(2)} + \dots + (x - x_0)(x - x_1)\dots(x - x_{n-1})f_i^{(n)}} \quad (4.65)$$

$P_n(x)$ is clearly a polynomial of degree n . To demonstrate that $P_n(x)$ passes exactly through the data points, let's substitute the data points into Eq. (4.65). Thus,

$$P_n(x_0) = f_i^{(0)} + (0)f_i^{(1)} + \dots + f_i^{(0)} = f_0 \quad (4.66)$$

$$P_n(x_1) = f_i^{(0)} + (x_1 - x_0)f_i^{(1)} + (x_1 - x_0)(0)f_i^{(2)} + \dots \quad (4.67a)$$

$$P_n(x_1) = f_0 + (x_1 - x_0) \frac{(f_1 - f_0)}{(x_1 - x_0)} = f_0 + (f_1 - f_0) = f_1 \quad (4.67b)$$

$$P_n(x_2) = f_i^{(0)} + (x_2 - x_0)f_i^{(1)} + (x_2 - x_0)(x_2 - x_1)f_i^{(2)} + (x_2 - x_0)(x_2 - x_1)(0)f_i^{(3)} + \dots \quad (4.68a)$$

$$P_n(x_2) = f_0 + (x_2 - x_0) \frac{(f_1 - f_0)}{(x_1 - x_0)} + (x_2 - x_0)(x_2 - x_1) \frac{(f_2 - f_1)/(x_2 - x_1) - (f_1 - f_0)/(x_1 - x_0)}{(x_2 - x_0)} \quad (4.68b)$$

$$P_n(x_2) = f_0 + (f_2 - f_1) + (f_1 - f_0) = f_2 \quad \text{etc.} \quad (4.68c)$$

Since $P_n(x)$ is a polynomial of degree n and passes exactly through the $n + 1$ data points, it is obviously one form of the unique polynomial passing through the data points.

Example 4.6. Divided difference polynomials.

Consider the divided difference table presented in Example 4.5. Let's interpolate for $f(3.44)$ using the divided difference polynomial, Eq. (4.65), using $x_0 = 3.35$ as the base point. The exact solution is $f(3.44) = 1/3.44 = 0.290698$. From Eq. (4.65):

$$\begin{aligned} P_n(3.44) &= f_0^{(0)} + (3.44 - x_0)f_0^{(1)} + (3.44 - x_0)(3.44 - x_1)f_0^{(2)} \\ &\quad + (3.44 - x_0)(3.44 - x_1)(3.44 - x_2)f_0^{(3)} \end{aligned} \quad (4.69)$$

Substituting the values of x_0 to x_2 and $f_0^{(0)}$ to $f_0^{(3)}$ into Eq. (4.69) gives

$$\begin{aligned} P_n(3.44) &= 0.298507 + (3.44 - 3.35)(-0.087780) \\ &\quad + (3.44 - 3.35)(3.44 - 3.4)(0.024933) \\ &\quad + (3.44 - 3.35)(3.44 - 3.4)(3.44 - 3.5)(-0.006132) \end{aligned} \quad (4.70)$$

Evaluating Eq. (4.70) term by term gives

$$P_n(3.44) = 0.298507 - 0.007900 + 0.000089 + 0.000001 \quad (4.71)$$

Summing the terms yields the following results and errors:

$P(3.44) = 0.290607$	linear interpolation	$\text{Error}(3.44) = -0.000091$
$= 0.290696$	quadratic interpolation	$= -0.000002$
$= 0.290697$	cubic interpolation	$= -0.000001$

The advantage of higher-degree interpolation is obvious.

The above results are not the most accurate possible since the data points in Table 4.4 are in monotonic order, which make the linear interpolation result actually linear extrapolation. Rearranging the data in order of closeness to $x = 3.44$ yields the results presented in Table 4.5. From Eq. (4.65):

$$\begin{aligned} P_n(3.44) &= 0.294118 + (3.44 - 3.40)(-0.084040) \\ &\quad + (3.44 - 3.40)(3.44 - 3.50)(0.024940) \\ &\quad + (3.44 - 3.40)(3.44 - 3.50)(3.44 - 3.35)(-0.006150) \end{aligned} \quad (4.72)$$

Evaluating Eq. (4.72) term by term gives

$$P_n(3.44) = 0.294118 - 0.003362 - 0.000060 + 0.000001 \quad (4.73)$$

Table 4.5. Rearranged Divided Difference Table

x_i	$f_i^{(0)}$	$f_i^{(1)}$	$f_i^{(2)}$	$f_i^{(3)}$
3.40	0.294118			
3.50	0.285714	-0.084040	0.024940	
3.35	0.298507	-0.085287	0.023710	-0.006150
3.60	0.277778	-0.082916		

Summing the terms yields the following results and errors:

$$\begin{array}{lll}
 P(3.44) = 0.290756 & \text{linear interpolation} & \text{Error} = 0.000058 \\
 & = 0.290697 & = -0.000001 \\
 & = 0.290698 & = 0.000000
 \end{array}$$

The linear interpolation value is much more accurate due to the centering of the data. The quadratic and cubic interpolation values are the same as before, except for round-off errors, because the same points are used in those two interpolations. These results are the same as the results obtained in the previous examples.

4.6 DIFFERENCE TABLES AND DIFFERENCE POLYNOMIALS

Fitting approximating polynomials to tabular data is considerably simpler when the values of the independent variable are equally spaced. Implementation of polynomial fitting for equally spaced data is best accomplished in terms of differences. Consequently, the concept of differences, difference tables, and difference polynomials are introduced in this section.

4.6.1. Difference Tables

A difference table is an arrangement of a set of data, $[x, f(x)]$, in a table with the x values in monotonic ascending order, with additional columns composed of the differences of the numbers in the preceding column. A triangular array is obtained, as illustrated in Table 4.6.

The numbers appearing in a difference table are unique. However, three different interpretations can be assigned to these numbers, each with its unique notation. The forward difference relative to point i is $(f_{i+1} - f_i)$, the backward difference relative to point $i + 1$ is $(f_{i+1} - f_i)$, and the centered difference relative to point $i + 1/2$ is $(f_{i+1} - f_i)$. The forward difference operator Δ is defined as

$$\Delta f(x_i) = \Delta f_i = (f_{i+1} - f_i) \quad (4.74)$$

The backward difference operator ∇ is defined as

$$\nabla f(x_{i+1}) = \nabla f_{i+1} = (f_{i+1} - f_i) \quad (4.75)$$

The centered difference operator δ is defined as

$$\delta f(x_{i+1/2}) = \delta f_{i+1/2} = (f_{i+1} - f_i) \quad (4.76)$$

A difference table, such as Table 4.6, can be interpreted as a forward-difference table, a backward-difference table, or a centered-difference table, as illustrated in Figure

Table 4.6. Table of Differences

x	$f(x)$			
x_0	f_0			
x_1	f_1	$(f_1 - f_0)$	$(f_2 - 2f_1 + f_0)$	
x_2	f_2	$(f_2 - f_1)$	$(f_3 - 2f_2 + f_1)$	$(f_3 - 3f_2 + 3f_1 - f_0)$
x_3	f_3	$(f_3 - f_2)$		

x	f	Δf	$\Delta^2 f$	$\Delta^3 f$
x_0	f_0			
x_1	f_1	Δf_0	$\Delta^2 f_0$	
x_2	f_2	Δf_1	$\Delta^2 f_1$	$\Delta^3 f_0$
x_3	f_3	Δf_2		

x	f	∇f	$\nabla^2 f$	$\nabla^3 f$
x_{-3}	f_{-3}			
x_{-2}	f_{-2}	∇f_{-2}		
x_{-1}	f_{-1}	∇f_{-1}	$\nabla^2 f_{-1}$	
x_0	f_0	∇f_0	$\nabla^2 f_0$	$\nabla^3 f_0$

x	f	δf	$\delta^2 f$	$\delta^3 f$
x_{-1}	f_{-1}			
x_0	f_0	$\delta f_{-1/2}$	$\delta^2 f_0$	
x_1	f_1	$\delta f_{1/2}$	$\delta^2 f_1$	$\delta^3 f_{1/2}$
x_2	f_2	$\delta f_{3/2}$		

Figure 4.7 (a) Forward-difference table. (b) Backward-difference table. (c) Centered-difference table.

4.7. The numbers in the tables are identical. Only the notation is different. The three different types of interpretation and notation simplify the use of difference tables in the construction of approximating polynomials, which is discussed in Sections 4.6.2 to 4.6.4.

Example 4.7. Difference table.

Let's construct a six-place difference table for the function $f(x) = 1/x$ for $3.1 \leq x \leq 3.9$ with $\Delta x = 0.1$. The results are presented in Table 4.7, which uses the forward-difference notation to denote the columns of differences.

Table 4.7. Difference Table

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$	$\Delta^4 f(x)$	$\Delta^5 f(x)$
3.1	0.322581					
3.2	0.312500	-0.010081	0.000611			
3.3	0.303030	-0.009470	0.000558	-0.000053	0.000003	
3.4	0.294118	-0.008912	0.000508	-0.000050	0.000010	0.000007
3.5	0.285714	-0.008404	0.000468	-0.000040	0.000000	-0.000010
3.6	0.277778	-0.007936	0.000428	-0.000040	0.000008	0.000008
3.7	0.270270	-0.007508	0.000396	-0.000032	0.000000	-0.000008
3.8	0.263158	-0.007112	0.000364	-0.000032		
3.9	0.256410	-0.006748				

Several observations can be made from Table 4.7. The first and second differences are quite smooth. The third differences, while monotonic, are not very smooth. The fourth differences are not monotonic, and the fifth differences are extremely ragged. The magnitudes of the higher-order differences decrease rapidly. If the differences are not smooth and decreasing, several possible explanations exist:

1. The original data set has errors.
2. The increment Δx may be too large.
3. There may be a singularity in $f(x)$ or its derivatives in the range of the table.

Difference tables are useful for evaluating the quality of a set of tabular data.

Tabular data have a finite number of digits. The last digit is typically rounded off. Round-off has an effect on the accuracy of the higher-order differences. To illustrate this effect, consider a difference table showing only the round-off error in the last significant digit. The worst possible round-off situation occurs when every other number is rounded off by one-half in opposing directions, as illustrated in Table 4.8. Table 4.8 shows that the errors due to round-off in the original data oscillate and double in magnitude for each higher-order difference. The maximum error in the differences is given by

$$\boxed{\text{Maximum round-off error in } \Delta^n f = \pm 2^{n-1}} \quad (4.77)$$

For the results presented in Table 4.7, $\Delta^5 f$ oscillates between -10 and $+8$. From Eq. (4.77), the maximum round-off error in $\Delta^5 f$ is $\pm 2^{5-1} = \pm 16$. Consequently, the $\Delta^5 f$ values are completely masked by the accumulated round-off error.

Polynomial fitting can be accomplished using the values in a difference table. The degree of polynomial needed to give a satisfactory fit to a set of tabular data can be estimated by considering the properties of polynomials. The n th-degree polynomial $P_n(x)$ is given by

$$P_n(x) = a_n x^n + (\text{Lower Degree Terms}) = a_n x^n + (\text{LDTs}) \quad (4.78)$$

In Section 4.2, it is shown that [see Eqs. (4.11n) and (4.12)]

$$P_n^{(n)}(x) = n! a_n = \text{constant} \quad (4.79)$$

$$P_n^{(n+1)}(x) = 0 \quad (4.80)$$

Table 4.8. Difference Table of Round-off Errors

x	f	Δf	$\Delta^2 f$	$\Delta^3 f$	$\Delta^4 f$
-	+1/2				
-	-1/2	-1	2		
-	+1/2	1	-2	-4	8
-	-1/2	-1	2	4	-8
-	+1/2	1	-2	-4	
-	-1/2	-1			

Let's evaluate the first forward difference of $P_n(x)$:

$$\Delta[P_n(x)] = \Delta(a_n x^n) + \Delta(\text{LDTs}) \quad (4.81)$$

$$\Delta[P_n(x)] = a_n(x+h)^n - a_n x^n + (\text{LDTs}) \quad (4.82)$$

Expanding $(x+h)^n$ in a binomial expansion gives

$$\Delta[P_n(x)] = [a_n x^n + a_n n x^{n-1} h + \cdots + a_n h^n] - a_n x^n + (\text{LDTs}) \quad (4.83)$$

which yields

$$\Delta[P_n(x)] = a_n n h x^{n-1} + (\text{LDTs}) = P_{n-1}(x) \quad (4.84)$$

Evaluating the second forward difference of $P_n(x)$ gives

$$\Delta^2[P_n(x)] = \Delta[(a_n n h)x^{n-1}] + \Delta(\text{LDTs}) \quad (4.85)$$

which yields

$$\Delta^2[P_n(x)] = a_n n(n-1)h^2 x^{n-2} + (\text{LDTs}) \quad (4.86)$$

In a similar manner it can be shown that

$$\Delta^n P_n(x) = a_n n! h^n = \text{constant} \quad \text{and} \quad \Delta^{n+1} P_n(x) = 0 \quad (4.87)$$

Note the similarity between $P_n^{(n)}(x)$ and $\Delta^n P_n(x)$. In fact, $P_n^{(n)}(x) = \Delta^n P_n(x)/h^n$. Thus, if $f(x) = P_n(x)$, then $\Delta^n f(x) = \text{constant}$. Consequently, if $\Delta^n f(x) \approx \text{constant}$, $f(x)$ can be approximated by $P_n(x)$.

4.6.2. The Newton Forward-Difference Polynomial

Given $n+1$ data points, $[x, f(x)]$, one form of the unique n th-degree polynomial that passes through the $n+1$ points is given by

$$P_n(x) = f_0 + s \Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!} \Delta^3 f_0 \\ + \cdots + \frac{s(s-1)(s-2) \cdots [s-(n-1)]}{n!} \Delta^n f_0 \quad (4.88)$$

where s is the interpolating variable

$$s = \frac{x - x_0}{\Delta x} = \frac{x - x_0}{h} \quad \text{and} \quad x = x_0 + sh \quad (4.89)$$

Equation (4.88) does not look anything like the direct fit polynomial [see Eq. (4.34)], the Lagrange polynomial [see Eq. (4.46)], or the divided difference polynomial [see Eq. (4.65)]. However, if Eq. (4.88) is a polynomial of degree n and passes exactly through the $n+1$ data points, it must be one form of the unique polynomial that passes through this set of data.

The interpolating variable, $s = (x - x_0)/h$, is linear in x . Consequently, the last term in Eq. (4.88) is order n , and Eq. (4.88) is an n th-degree polynomial. Let $s = 0$. Then $x = x_0$, $f = f_0$, and $P_n(x_0) = f_0$. Let $s = 1$. Then $x = x_0 + h = x_1$, $f = f_1$, and $P_n(x_1) = f_0 + \Delta f_0 = f_0 + (f_1 - f_0) = f_1$. In a similar manner, it can be shown that $P_n(x) = f(x)$ for the $n+1$ discrete points. Therefore, $P_n(x)$ is the desired unique n th-degree polynomial. Equation (4.88) is called the *Newton forward-difference polynomial*.

The Newton forward-difference polynomial can be expressed in a more compact form by introducing the definition of the *binomial coefficient*. Thus,

$$\binom{s}{i} = \frac{s(s-1)(s-2)\cdots(s-[i-1])}{i!} \quad (4.90)$$

In terms of binomial coefficients, the Newton forward-difference polynomial is

$$P_n(x) = f_0 + \binom{s}{1}\Delta f_0 + \binom{s}{2}\Delta^2 f_0 + \binom{s}{3}\Delta^3 f_0 + \dots \quad (4.91)$$

A major advantage of the Newton forward-difference polynomial, in addition to its simplicity, is that each higher-degree polynomial is obtained from the previous lower-degree polynomial simply by adding the next term. The work already performed for the lower-degree polynomial does not have to be repeated. This feature is in sharp contrast to the direct fit polynomial and the Lagrange polynomial, where all of the work must be repeated each time the degree of the polynomial is changed. This feature makes it simple to determine when the desired accuracy has been obtained. When the next term in the polynomial is less than some prespecified value, the desired accuracy has been obtained.

Example 4.8. Newton forward-difference polynomial.

From the six-place difference table for $f(x) = 1/x$, Table 4.7, calculate $P(3.44)$ by the Newton forward-difference polynomial. The exact solution is $f(3.44) = 1/3.44 = 0.290698\dots$. In Table 4.7, $h = 0.1$. Choose $x_0 = 3.40$. Then,

$$s = \frac{x - x_0}{h} = \frac{3.44 - 3.40}{0.1} = 0.4 \quad (4.92)$$

Equation (4.88) gives

$$P(3.44) = f(3.4) + s\Delta f(3.4) + \frac{s(s-1)}{2!}\Delta^2 f(3.4) + \frac{s(s-1)(s-2)}{3!}\Delta^3 f(3.4) + \dots \quad (4.93)$$

Substituting $s = 0.4$ and the values of the differences from Table 4.7 into Eq. (4.93) gives

$$\begin{aligned} P(3.44) &= 0.294118 + (0.4)(-0.008404) + (0.4)\frac{(0.4-1)}{2}(0.000468) \\ &\quad + \frac{(0.4)(0.4-1)(0.4-2)}{6}(-0.000040) + \dots \end{aligned} \quad (4.94)$$

Evaluating Eq. (4.94) term by term yields the following results and errors:

$$\begin{array}{lll} P(3.44) = 0.290756 & \text{linear interpolation} & \text{Error}(3.44) = 0.000058 \\ = 0.290700 & \text{quadratic interpolation} & = 0.000002 \\ = 0.290698 & \text{cubic interpolation} & = 0.000000 \end{array}$$

The advantage of higher-degree interpolation is obvious.

In this example, the base point, $x_0 = 3.4$, was selected so that the point of interpolation, $x = 3.44$, falls within the range of data used to determine the polynomial, that is, interpolation occurs. If x_0 is chosen so that x does not fall within the range of fit,

extrapolation occurs, and the results are less accurate. For example, let $x_0 = 3.2$, for which $s = 2.4$. The following results and errors are obtained:

$P(3.44) = 0.289772$	linear extrapolation	Error = -0.000926
= 0.290709	quadratic extrapolation	= 0.000011
= 0.290698	cubic interpolation	= 0.000000

The increase in error is significant for linear and quadratic extrapolation. For $x_0 = 3.2$, the cubic yields an interpolating polynomial.

The error term for the Newton forward-difference polynomial can be obtained from the general error term [see Eq. (4.9)]:

$$\text{Error}(x) = \frac{1}{(n+1)!} (x - x_0)(x - x_1) \cdots (x - x_n) f^{(n+1)}(\xi) \quad (4.95)$$

From Eq. (4.89),

$$(x - x_0) = (x_0 + sh) - x_0 = sh \quad (4.96a)$$

$$(x - x_1) = (x_0 + sh) - x_1 = sh - (x_1 - x_0) = (s - 1)h \quad (4.96b)$$

$$\dots \dots \dots \quad (4.96n)$$

$$(x - x_n) = (x_0 + sh) - x_n = sh - (x_n - x_0) = (s - n)h$$

Substituting Eq. (4.96) into Eq. (4.95) gives

$$\text{Error}(x) = \frac{1}{(n+1)!} s(s-1)(s-2) \cdots (s-n) h^{n+1} f^{(n+1)}(\xi) \quad (4.97)$$

which can be written as

$$\text{Error}(x) = \binom{s}{n+1} h^{n+1} f^{(n+1)}(\xi) \quad (4.98)$$

From Eq. (4.91), for $P_n(x)$, the term after the n th-term is

$$\binom{s}{n+1} \Delta^{n+1} f_0 \quad (4.99)$$

The error term, Eq. (4.98), can be obtained from Eq. (4.99) by the replacement

$$\Delta^{n+1} f_0 \rightarrow h^{n+1} f^{(n+1)}(\xi) \quad (4.100)$$

This procedure can be used to obtain the error term for all polynomials based on a set of discrete data points.

4.6.3 The Newton Backward-Difference Polynomial

The Newton forward-difference polynomial, Eq. (4.88), can be applied at the top or in the middle of a set of tabular data, where the downward-sloping forward differences illustrated in Figure 4.7a exist. However, at the bottom of a set of tabular data, the required forward differences do not exist, and the Newton forward-difference polynomial cannot be used. In that case, an approach that uses the upward-sloping backward differences illustrated in Figure 4.7b is required. Such a polynomial is developed in this section.

Given $n + 1$ data points, $[x, f(x)]$, one form of the unique n th-degree polynomial that passes through the $n + 1$ points is given by

$$\boxed{P_n(x) = f_0 + s \nabla f_0 + \frac{s(s+1)}{2!} \nabla^2 f_0 + \frac{s(s+1)(s+2)}{3!} \nabla^3 f_0 + \cdots + \frac{s(s+1) \cdots [s+(n-1)]}{n!} \nabla^n f_0} \quad (4.101)$$

where s is the interpolating variable

$$s = \frac{x - x_0}{\Delta x} = \frac{x - x_0}{h} \quad \text{and} \quad x = x_0 + sh \quad (4.102)$$

The interpolating variable, $s = (x - x_0)/h$, is linear in x . Consequently, the last term in Eq. (4.101) is order n , and Eq. (4.101) is an n th-degree polynomial. Let $s = 0$. Then $x = x_0$, $f = f_0$, and $P_n(x_0) = f_0$. Let $s = -1$. Then $x = x_0 - h = x_{-1}$, $f = f_{-1}$, and $P_n(x_{-1}) = f_0 - \nabla f_0 = f_0 - (f_0 - f_{-1}) = f_{-1}$. In a similar manner, it can be shown that $P_n(x) = f(x)$ for the $n + 1$ discrete points. Therefore, $P_n(x)$ is the desired unique n th-degree polynomial. Equation (4.101) is called the *Newton backward-difference polynomial*.

The Newton backward-difference polynomial can be expressed in a more compact form using a nonstandard definition of the binomial coefficient. Thus,

$$\binom{s^+}{i} = \frac{s(s+1)(s+2) \cdots (s+[i-1])}{i!} \quad (4.103)$$

In terms of the *nonstandard binomial coefficients*, the Newton backward-difference polynomial is

$$\boxed{P_n(x) = f_0 + \binom{s^+}{1} \nabla f_0 + \binom{s^+}{2} \nabla^2 f_0 + \binom{s^+}{3} \nabla^3 f_0 + \cdots} \quad (4.104)$$

Example 4.9. Newton backward-difference polynomial.

From the six-place difference table for $f(x) = 1/x$, Table 4.7, calculate $P(3.44)$ by the Newton backward-difference polynomial. The exact solution is $f(3.44) = 0.290698 \dots$. In Table 4.7, $h = 0.1$. Choose $x_0 = 3.50$. Then,

$$s = \frac{x - x_0}{h} = \frac{3.44 - 3.50}{0.1} = -0.6 \quad (4.105)$$

Equation (4.101) gives

$$P(3.44) = f(3.5) + s \nabla f(3.5) + \frac{s(s+1)}{2!} \nabla^2 f(3.5) + \frac{s(s+1)(s+2)}{3!} \nabla^3 f(3.5) + \cdots \quad (4.106)$$

Substituting $s = -0.6$ and the values of the differences from Table 4.7 into Eq. (4.106) gives

$$\begin{aligned} P(3.44) &= 0.285714 + (-0.6)(-0.008404) + \frac{(-0.6)(-0.6+1)}{2}(0.000508) \\ &\quad + \frac{(-0.6)(-0.6+1)(-0.6+2)}{6}(-0.000050) + \dots \end{aligned} \quad (4.107)$$

Evaluating Eq. (4.107) term by term yields the following results and errors:

$P(3.44) = 0.290756$	linear interpolation	Error = 0.000058
= 0.290695	quadratic interpolation	= -0.000003
= 0.290698	cubic interpolation	= 0.000000

The advantages of higher-degree interpolation are obvious.

The error term for the Newton backward-difference polynomial can be obtained from the general error term, Eq. (4.9), by making the following substitutions:

$$(x - x_0) = sh \quad (4.108a)$$

$$(x - x_1) = (x_0 + sh) - x_1 = sh + (x_0 - x_1) = (s + 1)h \quad (4.108b)$$

.....

$$(x - x_n) = (x_0 + sh) - x_n = sh + (x_0 - x_n) = (s + n)h \quad (4.108n)$$

Substituting Eq. (4.108) into Eq. (4.9) yields

$$\text{Error}(x) = \frac{1}{(n+1)!} s(s+1)(s+2) \cdots (s+n-1)(s+n) h^{n+1} f^{(n+1)}(\xi) \quad (4.109)$$

which can be written as

$$\text{Error}(x) = \binom{s^+}{n+1} h^{n+1} f^{(n+1)}(\xi) \quad (4.110)$$

Equation (4.110) can be obtained from Eq. (4.104) by the following replacement in the $(n+1)$ st term:

$$\nabla^{n+1} f_0 \rightarrow h^{n+1} f^{(n+1)}(\xi) \quad (4.111)$$

4.6.4. Other Difference Polynomials

The Newton forward-difference polynomial and the Newton-backward-difference polynomial presented in Sections 4.6.2 and 4.6.3, respectively, are examples of approximating polynomials based on differences. The Newton forward-difference polynomial uses forward differences, which follow a downward-sloping path in the forward-difference table illustrated in Figure 4.7a. The Newton backward-difference polynomial uses backward differences, which follow an upward-sloping path in the backward-difference table illustrated in Figure 4.7b. Numerous other difference polynomials based on other paths through a difference table can be constructed. Two of the more important ones are presented in this section.

The Newton forward-difference and backward-difference polynomials are essential for fitting an approximating polynomial at the beginning and end, respectively, of a set of

tabular data. However, other forms of difference polynomials can be developed in the middle of a set of tabular data by using the centered-difference table presented in Figure 4.7c. The base point for the *Stirling centered-difference polynomial* is point x_0 . This polynomial is based on the values of the centered differences with respect to x_0 . That is, the polynomial follows a horizontal path through the difference table which is centered on point x_0 . As illustrated in Figure 4.7c, even centered differences with respect to x_0 exist, but odd centered differences do not. Odd centered differences are based on the averages of the centered differences at the half points $x_{-1/2}$ and $x_{1/2}$. The Stirling centered-difference polynomial is

$$\begin{aligned} P_n(x) = f_0 + \binom{s}{1} \frac{1}{2} (\delta f_{1/2} + \delta f_{-1/2}) + \frac{1}{2} \left[\binom{s+1}{2} + \binom{s}{2} \right] \delta^2 f_0 \\ + \binom{s+1}{3} \frac{1}{2} (\delta^3 f_{1/2} + \delta^3 f_{-1/2}) + \frac{1}{2} \left[\binom{s+2}{4} + \binom{s+1}{4} \right] \delta^4 f_0 \\ + \dots \end{aligned} \quad (4.112)$$

It can be shown by direct substitution that the Stirling centered-difference polynomials of even degree are order n and pass exactly through the data points used to construct the differences appearing in the polynomial. The odd-degree polynomials use data from one additional point.

The base point for the *Bessel centered-difference polynomial* is point $x_{1/2}$. This polynomial is based on the values of the centered differences with respect to $x_{1/2}$. That is, the polynomial follows a horizontal path through the difference table which is centered on point $x_{1/2}$. As illustrated in Figure 4.7c, odd centered differences with respect to $x_{1/2}$ exist, but even centered differences do not. Even centered differences are based on the averages of the centered differences at points x_0 and x_1 . The Bessel centered-difference polynomial is:

$$\begin{aligned} P_n(x) = \frac{1}{2} (f_0 + f_1) + \frac{1}{2} \left[\binom{s}{1} + \binom{s-1}{1} \right] \delta f_{1/2} + \binom{s}{2} \frac{1}{2} (\delta^2 f_0 + \delta^2 f_1) \\ + \frac{1}{2} \left[\binom{s+1}{3} + \binom{s}{3} \right] \delta^3 f_{1/2} + \binom{s+1}{4} \frac{1}{2} (\delta^4 f_0 + \delta^4 f_1) + \dots \end{aligned} \quad (4.113)$$

It can be shown by direct substitution that the Bessel centered-difference polynomials of odd degree are order n and pass exactly through the data points used to construct the centered differences appearing in the polynomial. The even-degree polynomials use data from one additional point.

Centered-difference polynomials are useful in the middle of a set of tabular data where the centered differences exist to the full extent of the table. However, from the uniqueness theorem for polynomials (see Section 4.2), the polynomial of degree n that passes through a specific set of $n + 1$ points is unique. Thus, the Newton polynomials and the centered-difference polynomials are all equivalent when fit to the same data points. The Newton polynomials are somewhat simpler to evaluate. Consequently, when the exact number of data points to be fit by a polynomial is prespecified, the Newton polynomials are recommended.

4.7 INVERSE INTERPOLATION

Interpolation is the process of determining the value of the dependent variable f corresponding to a particular value of the independent variable x when the function $f(x)$ is described by a set of tabular data. *Inverse interpolation* is the process of determining the value of the independent variable x corresponding to a particular value of the dependent variable f . In other words, inverse interpolation is evaluation of the inverse function $x(f)$. Inverse interpolation can be accomplished by:

1. Fitting a polynomial to the inverse function $x(f)$
2. Solving a direct polynomial $f(x)$ iteratively for $x(f)$

Fitting a polynomial to the inverse function $x(f)$ appears to be the obvious approach. However, some problems may occur. The inverse function $x(f)$ may not resemble a polynomial. The values of f most certainly are not equally spaced. In such cases, a direct fit polynomial $f(x)$ may be preferred, even though it must be solved iteratively for $x(f)$, for example, by Newton's method.

Example 4.10. Inverse interpolation.

Consider the following set of tabular data, which corresponds to the function $f(x) = 1/x$:

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$
3.3	0.303030		
3.4	0.294118	-0.008912	
3.5	0.285714	-0.008404	0.000508

Let's find the value of x for which $f(x) = 0.30$. The exact solution is $x = 1/f(x) = 1/0.3 = 3.333333 \dots$

Let's evaluate the quadratic Lagrange polynomial, $x = x(f)$, for $f = 0.30$. Thus,

$$x = \frac{(0.30 - 0.294118)(0.30 - 0.285714)}{(0.303030 - 0.294118)(0.303030 - 0.285714)} \quad (3.3)$$

$$+ \frac{(0.30 - 0.303030)(0.30 - 0.285714)}{(0.294118 - 0.303030)(0.294118 - 0.285714)} \quad (3.4)$$

$$+ \frac{(0.30 - 0.303030)(0.30 - 0.294118)}{(0.285714 - 0.303030)(0.285714 - 0.294118)} \quad (4.114)$$

which yields $x = 3.333301$. The error is $\text{Error} = 3.333301 - 3.333333 = -0.000032$.

Alternatively, let's evaluate the quadratic Newton forward-difference polynomial for $f = f(x)$. Thus,

$$f(x) = f_0 + s \Delta f_0 + \frac{s(s-1)}{2} \Delta^2 f_0 \quad (4.115)$$

Substituting the values from the table into Eq. (4.115) yields

$$0.30 = 0.303030 + s(-0.008912) + \frac{s(s-1)}{2}(0.000508) \quad (4.116)$$

Simplifying Eq. (4.116) gives

$$0.000508s^2 - 0.018332s + 0.006060 = 0 \quad (4.117)$$

Solving Eq. (4.117) by the quadratic formula yields two solutions, $s = 0.333654$ and 35.752960 . The second root is obviously extraneous. Solving for x from the first root yields

$$x = x_0 + sh = 3.3 + (0.333654)(0.1) = 3.333365 \quad (4.118)$$

The error is $\text{Error} = 3.333365 - 3.333333 = 0.000032$.

4.8 MULTIVARIATE APPROXIMATION

All of the approximating polynomials discussed so far are single-variable, or univariate, polynomials, that is, the dependent variable is a function of a single independent variable: $y = f(x)$. Many problems arise in engineering and science where the dependent variable is a function of two or more independent variables, for example, $z = f(x, y)$ is a two-variable, or bivariate, function. Such functions in general are called *multivariate functions*. When multivariate functions are given by tabular data, multivariate approximation is required for interpolation, differentiation, and integration. Two exact fit procedures for multivariate approximation are presented in this section:

- 1 Successive univariate polynomial approximation
2. Direct multivariate polynomial approximation

Approximate fit procedures for multivariate polynomial approximation are discussed in Section 4.10.4.

4.8.1. Successive Univariate Polynomial Approximation

Consider the bivariate function $z = f(x, y)$. A set of tabular data is illustrated in Table 4.9.

Simply stated, successive univariate approximation first fits a set of univariate approximating functions, for example, polynomials, at each value of one of the independent variables. For example, at each value of y_i , fit the univariate polynomials $z_i(x) = z(y_i, x)$. Interpolation, differentiation, or integration is then performed on each of these univariate approximating functions to yield values of the desired operation at the specified value of the other independent variable. For example, $z_i(x^*) = z_i(y_i, x^*)$. A univariate approximating function is then fit to those results as a function of the first independent variable. For example, fit the univariate polynomial $z = z(y)$ to the values

Table 4.9. Bivariate Tabular Data

	x_1	x_2	x_3	x_4
y_1	z_{11}	z_{12}	z_{13}	z_{14}
y_2	z_{21}	z_{22}	z_{23}	z_{24}
y_3	z_{31}	z_{32}	z_{33}	z_{34}
y_4	z_{41}	z_{42}	z_{43}	z_{44}

$z_i(x^*) = z_i(y_i, x^*)$. The final process of interpolation, differentiation, or integration is then performed on that univariate approximating function. The approximating functions employed for successive univariate approximation can be of any functional form. Successive univariate polynomial approximation is generally used.

Example 4.11. Successive univariate quadratic interpolation.

Consider the following values of enthalpy, $h(P, T)$ Btu/lbm, from a table of properties of steam, Table 4.10.

Use successive univariate quadratic polynomial interpolation to calculate the enthalpy h at $P = 1225$ psia and $T = 1104$ F (the value from the steam tables is 1556.0 Btu/lbm).

First fit the quadratic polynomial

$$h(P_i, T) = a + bT + cT^2 \quad (4.119)$$

at each pressure level P_i , and interpolate for the values of h at $T = 1100$ F. At $P = 1150$ psia:

$$a + 800b + (800)^2c = 1380.4 \quad (4.120a)$$

$$a + 1000b + (1000)^2c = 1500.2 \quad (4.120b)$$

$$a + 1200b + (1200)^2c = 1614.5 \quad (4.120c)$$

Solving for a , b , and c by Gauss elimination yields

$$h(1150, T) = 846.20 + 0.72275T - 0.00006875T^2 \quad (4.121)$$

Evaluating Eq. (4.121) at $T = 1100$ gives $h(1150, 1100) = 1558.04$ Btu/lbm. In a similar manner, at $P = 1200$ psia, $h(1200, T) = 825.50 + 0.75725T - 0.00008375T^2$, and $h(1200, 1100) = 1557.14$ Btu/lbm. At $P = 1250$ psia, $h(1250, T) = 823.60 + 0.75350T - 0.00008000T^2$, and $h(1250, 1100) = 1555.65$ Btu/lbm.

Next fit the quadratic polynomial

$$h(P, 1100) = a + bP + cP^2 \quad (4.122)$$

at $T = 1100$ F. and interpolate for the value of h at $P = 1225$ psia.

$$a + 1150b + (1150)^2c = 1558.04 \quad (4.123a)$$

$$a + 1200b + (1200)^2c = 1557.14 \quad (4.123b)$$

$$a + 1250b + (1250)^2c = 1555.65 \quad (4.123c)$$

Table 4.10. Enthalpy of Steam

P , psia	T , F		
	800	1000	1200
1150	1380.4	1500.2	1614.5
1200	1377.7	1499.0	1613.6
1250	1375.2	1497.1	1612.6

Solving for a , b , and c by Gauss elimination gives

$$h(P, 1100) = 1416.59 + 0.258125P - 0.00011750P^2 \quad (4.124)$$

Evaluating Eq. (4.124) at $P = 1225$ yields $h(1225, 1100) = 1556.5$ Btu/lbm. The error in this result is Error = $1556.5 - 1556.0 = 0.5$ Btu/lbm.

4.8.2. Direct Multivariate Polynomial Approximation

Consider the bivariate function, $z = f(x, y)$, and the set of tabular data presented in Table 4.10. The tabular data can be fit by a multivariate polynomial of the form

$$z = f(x, y) = a + bx + cy + dxy + ex^2 + fy^2 + gx^2y + hxy^2 + ix^3 + jy^3 + \dots \quad (4.125)$$

The number of data points must equal the number of coefficients in the polynomial. A linear bivariate polynomial in x and y is obtained by including the first four terms in Eq. (4.125). The resulting polynomial is exactly equivalent to successive univariate linear polynomial approximation if the same four data points are used. A quadratic bivariate polynomial in x and y is obtained by including the first eight terms in Eq. (4.125). The number of terms in the approximating polynomial increases rapidly as the degree of approximation increases. This leads to ill-conditioned systems of linear equations for determining the coefficients. Consequently, multivariate high-degree approximation must be used with caution.

Example 4.12. Direct multivariate linear interpolation.

Let's solve the interpolation problem presented in Example 4.11 by direct multivariate linear interpolation. The form of the approximating polynomial is

$$h = a + bT + cP + dTP \quad (4.126)$$

Substituting the four data points that bracket $P = 1225$ psia and $T = 1100$ F into Eq. (4.126) gives

$$1499.0 = a + (1000)b + (1200)c + (1000)(1200)d \quad (4.127a)$$

$$1497.1 = a + (1000)b + (1250)c + (1000)(1250)d \quad (4.127b)$$

$$1613.6 = a + (1200)b + (1200)c + (1200)(1200)d \quad (4.127c)$$

$$1612.6 = a + (1200)b + (1250)c + (1200)(1250)d \quad (4.127d)$$

Solving for a , b , c , and d by Gauss elimination yields

$$h = 1079.60 + 0.4650T - 0.1280P + 0.0900 \times 10^{-3}TP \quad (4.128)$$

Substituting $T = 1100$ and $P = 1225$ into Eq. (4.128) gives $h(1100, 1225) = 1555.5$ Btu/lbm. The error in this result is Error = $1555.5 - 1556.0 = -0.5$ Btu/lbm. The advantage of this approach is that Eq. (4.128) can be evaluated for other values of (T, P) , if required, without reevaluating the polynomial coefficients.

4.9 CUBIC SPLINES

Several procedures for fitting approximating polynomials to a set of tabular data are presented in Sections 4.3 to 4.6. Problems can arise when a single high-degree polynomial is fit to a large number of points. High-degree polynomials would obviously pass through all the data points themselves, but they can oscillate wildly between data points due to round-off errors and overshoot. In such cases, lower-degree polynomials can be fit to subsets of the data points. If the lower-degree polynomials are independent of each other, a *piecewise approximation* is obtained. An alternate approach is to fit a lower-degree polynomial to connect each pair of data points and to require the set of lower-degree polynomials to be consistent with each other in some sense. This type of polynomial is called a *spline function*, or simply a *spline*.

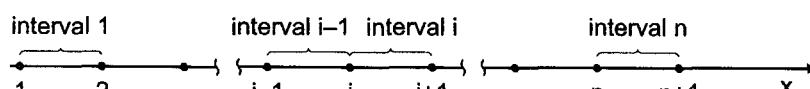
Splines can be of any degree. Linear splines are simply straight line segments connecting each pair of data points. Linear splines are independent of each other from interval to interval. Linear splines yield first-order approximating polynomials. The slopes (i.e., first derivatives) and curvature (i.e., second derivatives) are discontinuous at every data point. Quadratic splines yield second-order approximating polynomials. The slopes of the quadratic splines can be forced to be continuous at each data point, but the curvatures (i.e., the second derivatives) are still discontinuous.

A *cubic spline* yields a third-degree polynomial connecting each pair of data points. The slopes and curvatures of the cubic splines can be forced to be continuous at each data point. In fact, these requirements are necessary to obtain the additional conditions required to fit a cubic polynomial to two data points. Higher-degree splines can be defined in a similar manner. However, cubic splines have proven to be a good compromise between accuracy and complexity. Consequently, the concept of cubic splines is developed in this section.

The name *spline* comes from the thin flexible rod, called a spline, used by draftsmen to draw smooth curves through a series of discrete points. The spline is placed over the points and either weighted or pinned at each point. Due to the flexure properties of a flexible rod (typically of rectangular cross section), the slope and curvature of the rod are continuous at each point. A smooth curve is then traced along the rod, yielding a spline curve.

Figure 4.8 illustrates the discrete x space and defines the indexing convention. There are $n + 1$ total points, x_i ($i = 1, 2, \dots, n + 1$), n intervals, and $n - 1$ interior grid points, x_i ($i = 2, 3, \dots, n$). A cubic spline is to be fit to each interval. Thus,

$$f_i(x) = a_i + b_i x + c_i x^2 + d_i x^3 \quad (i = 1, 2, \dots, n) \quad (4.129)$$



- $n+1$ grid points, x_i ($i = 1, 2, \dots, n+1$)
- n intervals, $x_i \leq x \leq x_{i+1}$ ($i = 1, 2, \dots, n$)
- n cubic splines, $f_i(x)$ ($i = 1, 2, \dots, n$)
- $n-1$ interior grid points, x_i ($i = 2, 3, \dots, n$)

Figure 4.8 Cubic splines.

defines the cubic spline in interval i , $x_i \leq x \leq x_{i+1}$ ($i = 1, 2, \dots, n$). Since each cubic spline has four coefficients and there are n cubic splines, there are $4n$ coefficients to be determined. Thus, $4n$ boundary conditions, or constraints, must be available.

In the direct approach, the following constraints are applied.

1. The function values, $f(x_i) = f_i$ ($i = 2, 3, \dots, n$), must be the same in the two splines on either side of x_i at all of the $n - 1$ interior points. This constraint yields $2(n - 1)$ conditions.
2. The first derivative of the two splines on either side of point x_i must be equal at all of the $n - 1$ interior points. This constraint yields $(n - 1)$ conditions.
3. The second derivative of the two splines on either side of point x_i must be equal at all of the $n - 1$ interior points. This constraint yields $(n - 1)$ conditions.
4. The first and last spline must pass through the first (i.e., x_1) and last (i.e., x_{n+1}) points. That is, $f_1(x_1) = f_1$ and $f_n(x_{n+1}) = f_{n+1}$. This constraint yields 2 conditions.
5. The curvature [i.e., $f''(x)$] must be specified at the first (i.e., x_1) and last (i.e., x_{n+1}) points. That is, $f_1''(x_1) = f_1''$ and $f_n''(x_{n+1}) = f_{n+1}''$. This constraint yields 2 conditions.

When all of the conditions given above are assembled, $4n$ linear algebraic equations are obtained for the $4n$ spline coefficients a_i , b_i , c_i , and d_i ($i = 1, 2, \dots, n$). This set of equations can be solved by Gauss elimination. However, simpler approaches exist for determining cubic splines. Gerald and Wheatley (1999), Chapra and Canale (1998), and Press et al. (1989) present three such approaches. The approach presented by Chapra and Canale is followed below.

From the cubic equation, Eq. (4.129), it is obvious that the second derivative within each interval, $f_i''(x)$, is a linear function of x . The first-order Lagrange polynomial for the second derivative $f_i''(x)$ in interval i , $x_i \leq x \leq x_{i+1}$ ($i = 1, 2, \dots, n$), is given by

$$f_i''(x) = \frac{x - x_{i+1}}{x_i - x_{i+1}} f_i'' + \frac{x - x_i}{x_{i+1} - x_i} f_{i+1}'' \quad (4.130)$$

Integrating Eq. (4.130) twice yields expressions for $f_i'(x)$ and $f_i(x)$. Thus,

$$f_i'(x) = \frac{x^2/2 - xx_{i+1}}{x_i - x_{i+1}} f_i'' + \frac{x^2/2 - xx_i}{x_{i+1} - x_i} f_{i+1}'' + C \quad (4.131)$$

$$f_i(x) = \frac{x^3/6 - x^2x_{i+1}/2}{x_i - x_{i+1}} f_i''(x) + \frac{x^3/6 - x^2x_i/2}{x_{i+1} - x_i} f_{i+1}'' + Cx + D \quad (4.132)$$

Evaluating Eq. (4.132) at x_i and x_{i+1} and combining the results to eliminate the constants of integration C and D gives

$$f_i(x) = \frac{f_i''}{6(x_{i+1} - x_i)} (x_{i+1} - x)^3 + \frac{f_{i+1}''}{6(x_{i+1} - x_i)} (x - x_i)^3$$

$$+ \left[\frac{f_i}{x_{i+1} - x_i} - \frac{f_i''(x_{i+1} - x_i)}{6} \right] (x_{i+1} - x)$$

$$+ \left[\frac{f_{i+1}}{x_{i+1} - x_i} - \frac{f_{i+1}''(x_{i+1} - x_i)}{6} \right] (x - x_i) \quad (4.133)$$

Equation (4.133) is the desired cubic spline for increment i expressed in terms of the two unknown second derivatives f_i'' and f_{i+1}'' .

An expression for the second derivatives at the interior grid points, $f_i''(i = 2, 3, \dots, n)$, can be obtained by setting $f'_{i-1}(x_i) = f'_i(x_i)$. An expression for $f'_i(x)$ can be obtained by differentiating Eq. (4.133). Applying that expression to intervals $i - 1$ and i and evaluating those results at $x = x_i''$ gives expressions for $f'_{i-1}(x_i)$ and $f'_i(x_i)$. Equating those expressions yields

$$(x_i - x_{i-1})f''_{i-1} + 2(x_{i+1} - x_{i-1})f''_i + (x_{i+1} - x_i)f''_{i+1} = 6\frac{f_{i+1} - f_i}{x_{i+1} - x_i} - 6\frac{f_i - f_{i-1}}{x_i - x_{i-1}} \quad (4.134)$$

Applying Eq. (4.134) at the $n - 1$ interior points gives $n - 1$ coupled equations for the $n + 1$ second derivatives, $f_i''(i = 1, 2, \dots, n + 1)$. Two more values of f_i'' are required to close the system of equations.

The two additional conditions are obtained by specifying the values of f_1'' and f_{n+1}'' . Several approaches are available for specifying these two values:

1. Specify f_1'' and f_{n+1}'' if they are known. Letting $f_1'' = 0$ and/or $f_{n+1}'' = 0$ specifies a natural spline.
2. Specify f'_1 and/or f'_{n+1} and use Eq. (4.131) to develop a relationship between f'_1 and/or f'_{n+1} and $f'_1, f_1'', f'_2, f_2'', \dots, f'_{n+1}, f_{n+1}''$, etc. This requires the evaluation of the constant of integration C , which is not developed in this book.
3. Let $f_1'' = f_2''$ and $f_{n+1}'' = f_n''$.
4. Extrapolate f_1'' and f_{n+1}'' from interior values of f''_i .

The first approach, letting $f_1'' = f_{n+1}'' = 0$, is the most commonly employed approach.

Example 4.13. Cubic splines.

Let's illustrate the cubic spline by applying it to the data in the following table, which is for the function $f(x) = e^x - x^3$.

i	x	$f(x)$	$f''(x)$
1	-0.50	0.731531	0.0
2	0.00	1.000000	
3	0.25	1.268400	
4	1.00	1.718282	0.0

There are $n + 1 = 4$ grid points, $n = 3$ intervals, $n = 3$ cubic splines to be fit, and $n - 1 = 2$ interior grid points at which f_i'' must be evaluated.

The first step is to determine the values of f_i'' at the four grid points, $i = 1$ to 4. Let's apply the natural spline boundary condition at $i = 1$ and $i = 4$. Thus, $f_1'' = f_4'' = 0.0$. Applying Eq. (4.134) at grid points 2 and 3 gives

$$i = 2: (x_2 - x_1)f''_1 + 2(x_3 - x_1)f''_2 + (x_3 - x_2)f''_3 = 6\frac{f_3 - f_2}{x_3 - x_2} - 6\frac{f_2 - f_1}{x_2 - x_1} \quad (4.135)$$

$$i = 3: (x_3 - x_2)f''_2 + 2(x_4 - x_2)f''_3 + (x_4 - x_3)f''_4 = 6\frac{f_4 - f_3}{x_4 - x_3} - 6\frac{f_3 - f_2}{x_3 - x_2} \quad (4.136)$$

Substituting values from the table into Eqs. (4.135) and (4.136) gives

$$(0.5)(0.0) + 2(0.75)f_2'' + (0.25)f_3'' = 6\left(\frac{0.268400}{0.25} - \frac{0.268469}{0.50}\right) \quad (4.137a)$$

$$(0.25)f_2 + 2(1.0)f_3'' + (0.75)(0.0) = 6\left(\frac{0.449882}{0.75} - \frac{0.268400}{0.25}\right) \quad (4.137b)$$

Evaluating Eqs. (4.137) gives

$$1.50f_2'' + 0.25f_3'' = 3.219972 \quad (4.138a)$$

$$0.25f_2'' + 2.0f_3'' = -2.842544 \quad (4.138b)$$

Solving Eq. (4.138) by Gauss elimination yields

$$f_2'' = 2.434240 \quad \text{and} \quad f_3'' = -1.725552 \quad (4.139)$$

The second step is to substitute the numerical values into Eq. (4.133) for the three intervals, (x_1, x_2) , (x_2, x_3) , and (x_3, x_4) , to determine the cubic splines for the three intervals. Thus, for interval 1, that is, (x_1, x_2) , Eq. (4.133) becomes

$$\begin{aligned} f_1(x) &= \frac{f_1''}{6(x_2 - x_1)}(x_2 - x)^3 + \frac{f_2''}{6(x_2 - x_1)}(x - x_1)^3 \\ &\quad + \left[\frac{f_1}{x_2 - x_1} - \frac{f_1''(x_2 - x_1)}{6} \right](x_2 - x) + \left[\frac{f_2}{x_2 - x_1} - \frac{f_2''(x_2 - x_1)}{6} \right](x - x_1) \end{aligned} \quad (4.140)$$

Similar equations are obtained for intervals 2 and 3. Substituting values into these three equations gives

$$\begin{aligned} f_1(x) &= (0.0) + \frac{2.434240}{6(0.5)}[x - (-0.5)]^3 + \left(\frac{0.731531}{0.5} - 0.0\right)(0.0 - x) \\ &\quad + \left[\frac{1.0}{0.5} - \frac{2.434240(0.5)}{6} \right][x - (-0.5)] \end{aligned} \quad (4.141a)$$

$$\begin{aligned} f_2(x) &= \frac{2.434240}{6(0.25)}(0.25 - x)^3 + \frac{-1.725552}{6(0.25)}(x - 0.0)^3 \\ &\quad + \left[\frac{1.0}{0.25} - \frac{2.434240(0.25)}{6} \right](0.25 - x) \\ &\quad + \left[\frac{1.268400}{0.25} - \frac{-1.725552(0.25)}{6} \right](x - 0.0) \end{aligned} \quad (4.141b)$$

$$\begin{aligned} f_3(x) &= \frac{-1.725552}{6(0.75)}(1.0 - x)^3 + (0.0) + \left[\frac{1.268400}{0.75} - \frac{-1.725552(0.75)}{6} \right](1.0 - x) \\ &\quad + \left[\frac{1.718282}{0.75} - \frac{(0.0)(0.75)}{6} \right](x - 0.25) \end{aligned} \quad (4.141c)$$

Evaluating Eq. (4.141) yields

$$f_1(x) = 0.811413(x + 0.5)^3 - 1.463062x + 1.797147(x + 0.5)^3 \quad (4.142a)$$

$$f_2(x) = 1.622827(0.25 - x)^3 - 1.150368x^3 + 3.898573(0.25 - x) + 5.145498x \quad (4.142b)$$

$$f_3(x) = -0.383456(1.0 - x)^3 + 1.906894(1.0 - x) + 2.291043(x - 0.25) \quad (4.142c)$$

Equation (4.142) can be verified by substituting the values in the table into the equations.

4.10 LEAST SQUARES APPROXIMATION

Section 4.1 discusses the need for approximating functions for sets of discrete data (e.g., for interpolation, differentiation, and integration), the desirable properties of approximating functions, and the benefits of using polynomials for approximating functions. For small sets of smooth data, exact fits such as presented in Sections 4.3 to 4.9 are desirable. However, for large sets of data and sets of rough data, approximate fits are desirable. Approximate polynomial fits are the subject of this section.

4.10.1 Introduction

An approximate fit yields a polynomial that passes through the set of points in the *best possible manner* without being required to pass exactly through any of the points. Several definitions of best possible manner exist. Consider the set of discrete points, $[x_i, Y(x_i)] = (x_i, Y_i)$, and the approximate polynomial $y(x)$ chosen to represent the set of discrete points, as illustrated in Figure 4.9. The discrete points do not fall on the approximating polynomial. The deviations (i.e., distances) of the points from the approximating function must be minimized in some manner. Some ambiguity is possible in the definition of the deviation. For example, if the values of the independent variable x_i

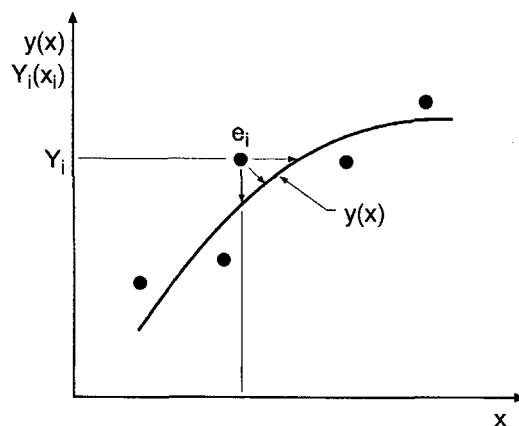


Figure 4.9 Approximate fit.

are considered exact, then all the deviation is assigned to the dependent variable Y_i , and the deviation e_i is the vertical distance between Y_i and $y_i = f(x_i)$. Thus,

$$e_i = Y_i - y_i \quad (4.143)$$

It is certainly possible that the values of Y_i are quite accurate, but the corresponding values of x_i are in error. In that case, the deviation would be measured by the horizontal distance illustrated in Figure 4.9. If x_i and Y_i both have uncertainties in their values, then the perpendicular distance between a point and the approximating function would be the deviation. The usual approach in approximate fitting of tabular data is to assume that the deviation is the vertical distance between a point and the approximating function, as specified by Eq. (4.143).

Several best fit criteria are illustrated in Figure 4.10 for a straight line approximation. Figure 4.10a illustrates the situation where the sum of the deviations at two points is minimized. Any straight line that passes through the midpoint of the line segment connecting the two points yields the sum of the deviations equal to zero. Minimizing the sum of the absolute values of the deviations would yield the unique line that passes exactly through the two points. That procedure also has deficiencies, however, as illustrated in Figure 4.10b, where two points having the same value of the independent variable have different values of the dependent variable. The best straight line obviously passes midway between these two points, but any line passing between these two points yields the same value for the sum of the absolute values of the deviations. The minimax criterion is illustrated in Figure 4.10c, where the maximum deviation is minimized. This procedure gives poor results when one point is far removed from the other points. Figure

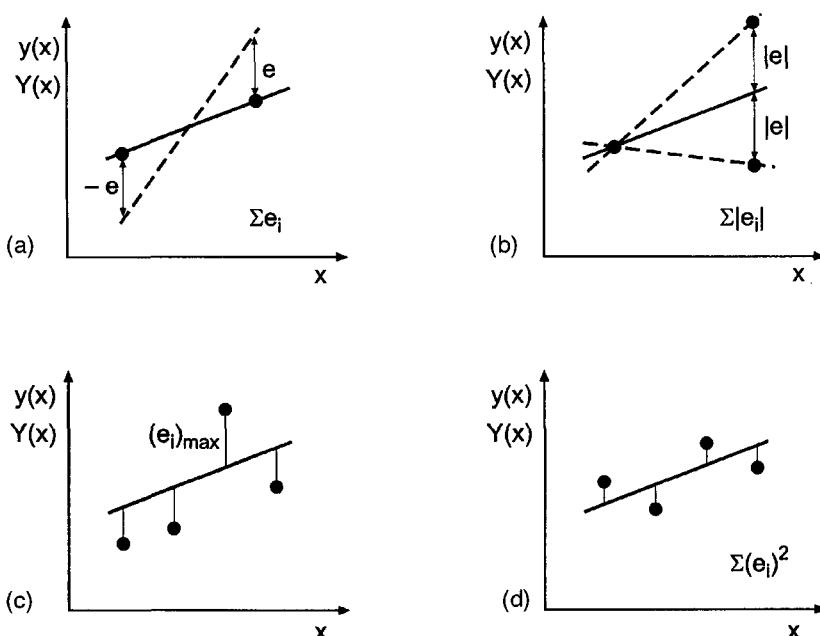


Figure 4.10 Best fit criteria. (a) Minimize $\sum e_i$. (b) Minimize $\sum |e_i|$. (c) Minimax. (d) Least squares.

4.10d illustrates the least squares criteria, in which the sum of the squares of the deviations is minimized. The least squares procedure yields a good compromise criterion for the best fit approximation.

The *least squares method* is defined as follows. Given N data points, $[x_i, Y(x_i)] = (x_i, Y_i)$, choose the functional form of the approximating function to be fit, $y = y(x)$, and minimize the sum of the squares of the deviations, $e_i = (Y_i - y_i)$.

4.10.2 The Straight Line Approximation

The simplest polynomial is a linear polynomial, the straight line. Least squares straight line approximations are an extremely useful and common approximate fit. The least squares straight line fit is determined as follows. Given N data points, (x_i, Y_i) , fit the best straight line through the set of data. The approximating function is

$$y = a + bx \quad (4.144)$$

At each value of x_i , Eq. (4.144) gives

$$y_i = a + bx_i \quad (i = 1, \dots, N) \quad (4.145)$$

The deviation e_i at each value of x_i is

$$e_i = Y_i - y_i \quad (i = 1, \dots, N) \quad (4.146)$$

The sum of the squares of the deviations defines the function $S(a, b)$:

$$S(a, b) = \sum_{i=1}^N (e_i)^2 = \sum_{i=1}^N (Y_i - a - bx_i)^2 \quad (4.147)$$

The function $S(a, b)$ is a minimum when $\partial S / \partial a = \partial S / \partial b = 0$. Thus,

$$\frac{\partial S}{\partial a} = \sum_{i=1}^N 2(Y_i - a - bx_i)(-1) = 0 \quad (4.148a)$$

$$\frac{\partial S}{\partial b} = \sum_{i=1}^N 2(Y_i - a - bx_i)(-x_i) = 0 \quad (4.148b)$$

Dividing Eqs. (4.148) by 2 and rearranging yields

$$aN + b \sum_{i=1}^N x_i = \sum_{i=1}^N Y_i \quad (4.149a)$$

$$a \sum_{i=1}^N x_i + b \sum_{i=1}^N x_i^2 = \sum_{i=1}^N x_i Y_i \quad (4.149b)$$

Equations (4.149) are called the *normal equations* of the least squares fit. They can be solved for a and b by Gauss elimination.

Example 4.14. Least squares straight line approximation.

Consider the constant pressure specific heat for air at low temperatures presented in Table 4.11, where T is the temperature (K) and C_p is the specific heat (J/gm-K). The exact values, approximate values from the least squares straight line approximation, and the

Table 4.11. Specific Heat of Air at Low Temperatures

T, K	C_p, exact	$C_p, \text{approx.}$	Error, %
300	1.0045	0.9948	-0.97
400	1.0134	1.0153	0.19
500	1.0296	1.0358	0.61
600	1.0507	1.0564	0.54
700	1.0743	1.0769	0.24
800	1.0984	1.0974	-0.09
900	1.1212	1.1180	-0.29
1000	1.1410	1.1385	-0.22

percent error are also presented in the table. Determine a least squares straight line approximation for the set of data:

$$C_p = a + bT \quad (4.150)$$

For this problem, Eq. (4.149) becomes

$$8a + b \sum_{i=1}^8 T_i = \sum_{i=1}^8 C_{p,i} \quad (4.151)$$

$$a \sum_{i=1}^8 T_i + b \sum_{i=1}^8 T_i^2 = \sum_{i=1}^8 T_i C_{p,i} \quad (4.152)$$

Evaluating the summations and substituting into Eqs. (4.151) and (4.152) gives

$$8a + 5200b = 8.5331 \quad (4.153a)$$

$$5200a + 3,800,000b = 5632.74 \quad (4.153b)$$

Solving for a and b by Gauss elimination without scaling or pivoting yields

$$C_p = 0.933194 + 0.205298 \times 10^{-3} T \quad (4.154)$$

Substituting the initial values of T into Eq. (4.154) gives the results presented in Table 4.11, which presents the exact data, the least squares straight line approximation, and the percent error. Figure 4.11 presents the exact data and the least squares straight line approximation. The straight line is not a very good approximation of the data.

4.10.3. Higher-Degree Polynomial Approximation

The least squares procedure developed in Section 4.10.2 can be applied to higher-degree polynomials. Given the N data points, (x_i, Y_i) , fit the best n th-degree polynomial through the set of data. Consider the n th-degree polynomial:

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \quad (4.155)$$

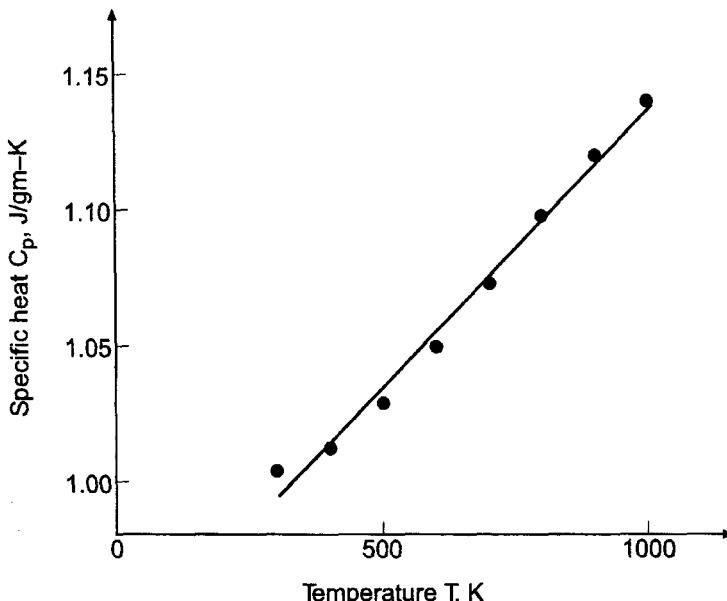


Figure 4.11 Least squares straight line approximation.

The sum of the squares of the deviations is given by

$$S(a_0, a_1, \dots, a_n) = \sum_{i=1}^N (e_i)^2 = \sum_{i=1}^N (Y_i - a_0 - a_1 x_i - \dots - a_n x_i^n)^2 \quad (4.156)$$

The function $S(a_0, a_1, \dots, a_n)$ is a minimum when

$$\frac{\partial S}{\partial a_0} = \sum_{i=1}^N 2(Y_i - a_0 - a_1 x_i - \dots - a_n x_i^n)(-1) = 0 \quad (4.157a)$$

.....

$$\frac{\partial S}{\partial a_n} = \sum_{i=1}^N 2(Y_i - a_0 - a_1 x_i - \dots - a_n x_i^n)(-x_i^n) = 0 \quad (4.157b)$$

Dividing Eqs. (4.157) by 2 and rearranging yields the normal equations:

$$a_0 N + a_1 \sum_{i=1}^N x_i + \dots + a_n \sum_{i=1}^N x_i^n = \sum_{i=1}^N Y_i \quad (4.158a)$$

.....

$$a_0 \sum_{i=1}^N x_i^n + a_1 \sum_{i=1}^N x_i^{n+1} + \dots + a_n \sum_{i=1}^N x_i^{2n} = \sum_{i=1}^N x_i^n Y_i \quad (4.158b)$$

Equation (4.158) can be solved for a_0 to a_n by Gauss elimination.

A problem arises for high-degree polynomials. The coefficients in Eq. (4.158), N to x_i^{2n} , can vary over a range of several orders of magnitude, which gives rise to ill-conditioned systems. Normalizing each equation helps the situation. Double precision

calculations are frequently required. Values of n up to 5 or 6 generally yield good results, values of n between 6 and 10 may or may not yield good results, and values of n greater than approximately 10 generally yield poor results.

Example 4.15. Least squares quadratic polynomial approximation.

Consider the constant pressure specific heat of air at high temperatures presented in Table 4.12, where T is the temperature (K) and C_p is the specific heat (J/gm-K). The exact values, approximate values from a least squares quadratic polynomial approximation, and the percent error are also presented in the table. Determine a least squares quadratic polynomial approximation for this set of data:

$$C_p = a + bT + cT^2 \quad (4.159)$$

For this problem, Eq. (4.158) becomes

$$5a + b \sum T_i + c \sum T_i^2 = \sum C_{p,i} \quad (4.160a)$$

$$a \sum T_i + b \sum T_i^2 + c \sum T_i^3 = \sum T_i C_{p,i} \quad (4.160b)$$

$$a \sum T_i^2 + b \sum T_i^3 + c \sum T_i^4 = \sum T_i^2 C_{p,i} \quad (4.160c)$$

Evaluating the summations and substituting into Eq. (4.160) gives

$$5a + 10 \times 10^3 b + 22.5 \times 10^6 c = 6.1762 \quad (4.161a)$$

$$10 \times 10^3 a + 22.5 \times 10^6 b + 55 \times 10^9 c = 12.5413 \times 10^3 \quad (4.161b)$$

$$22.5 \times 10^6 a + 55 \times 10^9 b + 142.125 \times 10^{12} c = 288.5186 \times 10^6 \quad (4.161c)$$

Solving for a , b , and c by Gauss elimination yields

$$C_p = 0.965460 + 0.211197 \times 10^{-3}T - 0.0339143 \times 10^{-6}T^2 \quad (4.162)$$

Substituting the initial values of T into Eq. (4.162) gives the results presented in Table 4.12. Figure 4.12 presents the exact data and the least squares quadratic polynomial approximation. The quadratic polynomial is a reasonable approximation of the discrete data.

Table 4.12. Specific Heat of Air at High Temperatures

T, K	$C_{p,exact}$	$C_{p,approx.}$	Error, %
1000	1.1410	1.1427	0.15
1500	1.2095	1.2059	-0.29
2000	1.2520	1.2522	0.02
2500	1.2782	1.2815	0.26
3000	1.2955	1.2938	-0.13

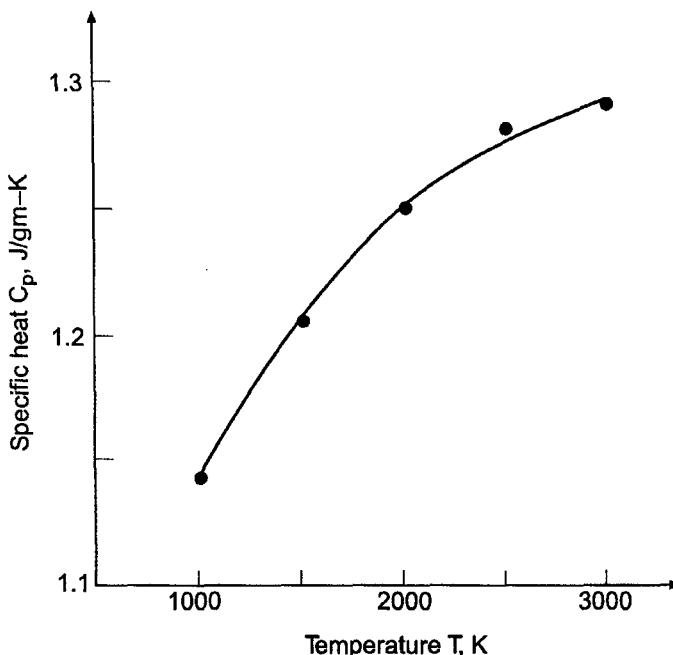


Figure 4.12 Least squares quadratic approximation.

4.10.4. Multivariate Polynomial Approximation

Many problems arise in engineering and science where the dependent variable is a function of two or more independent variables, for example, $z = f(x, y)$ is a two-variable, or bivariate, function. Two exact fit procedures for multivariate approximation are presented in Section 4.8. Least squares multivariate approximation is considered in this section.

Given the N data points, (x_i, y_i, Z_i) , fit the best linear bivariate polynomial through the set of data. Consider the linear polynomial:

$$z = a + bx + cy \quad (4.163)$$

The sum of the squares of the deviations is given by

$$S(a, b, c) = \sum (e_i)^2 = \sum (Z_i - a - bx_i - cy_i)^2 \quad (4.164)$$

The function $S(a, b, c)$ is a minimum when

$$\frac{\partial S}{\partial a} = \sum 2(Z_i - a - bx_i - cy_i)(-1) = 0 \quad (4.165a)$$

$$\frac{\partial S}{\partial b} = \sum 2(Z_i - a - bx_i - cy_i)(-x_i) = 0 \quad (4.165b)$$

$$\frac{\partial S}{\partial c} = \sum 2(Z_i - a - bx_i - cy_i)(-y_i) = 0 \quad (4.165c)$$

Dividing Eqs. (4.165) by 2 and rearranging yields the normal equations:

$$aN + b \sum x_i + c \sum y_i = \sum Z_i \quad (4.166a)$$

$$a \sum x_i + b \sum x_i^2 + c \sum x_i y_i = \sum x_i Z_i \quad (4.166b)$$

$$a \sum y_i + b \sum x_i y_i + c \sum y_i^2 = \sum y_i Z_i \quad (4.166c)$$

Equation (4.166) can be solved for a , b , and c by Gauss elimination.

A linear fit to a set of bivariate data may be inadequate. Consider the quadratic bivariate polynomial:

$$z = a + bx + cy + dx^2 + ey^2 + fxy \quad (4.167)$$

The sum of the squares of the deviations is given by

$$S(a, b, c, d, e, f) = \sum (Z_i - a - bx_i - cy_i - dx_i^2 - ey_i^2 - fx_i y_i)^2 \quad (4.168)$$

The function $S(a, b, \dots, f)$ is a minimum when

$$\frac{\partial S}{\partial a} = \sum 2(Z_i - a - bx_i - cy_i - dx_i^2 - ey_i^2 - fx_i y_i)(-1) = 0 \quad (4.169a)$$

$$\frac{\partial S}{\partial f} = \sum 2(Z_i - a - bx_i - cy_i - dx_i^2 - ey_i^2 - fx_i y_i)(-x_i y_i) = 0 \quad (4.169f)$$

Dividing Eqs. (4.169) by 2 and rearranging yields the normal equations:

$$aN + b \sum x_i + c \sum y_i + d \sum x_i^2 + e \sum y_i^2 + f \sum x_i y_i = \sum Z_i \quad (4.170a)$$

$$a \sum x_i + b \sum x_i^2 + c \sum x_i y_i + d \sum x_i^3 + e \sum x_i y_i^2 + f \sum x_i^2 y_i = \sum x_i Z_i \quad (4.170b)$$

$$a \sum y_i + b \sum x_i y_i + c \sum y_i^2 + d \sum x_i^2 y_i + e \sum y_i^3 + f \sum x_i y_i^2 = \sum y_i Z_i \quad (4.170c)$$

$$a \sum x_i^2 + b \sum x_i^3 + c \sum x_i^2 y_i + d \sum x_i^4 + e \sum x_i^2 y_i^2 + f \sum x_i^3 y_i = \sum x_i^2 Z_i \quad (4.170d)$$

$$a \sum y_i^2 + b \sum x_i y_i^2 + c \sum y_i^3 + d \sum x_i^2 y_i^2 + e \sum y_i^4 + f \sum x_i y_i^3 = \sum y_i^2 Z_i \quad (4.170e)$$

$$a \sum x_i y_i + b \sum x_i^2 y_i + c \sum x_i y_i^2 + d \sum x_i^3 y_i + e \sum x_i y_i^3 + f \sum x_i^2 y_i^2 = \sum x_i y_i Z_i \quad (4.170f)$$

Equation (4.170) can be solved for a to f by Gauss elimination.

Example 4.16. Least squares quadratic bivariate polynomial approximation.

Let's rework Example 4.12 to calculate the enthalpy of steam at $P = 1225$ psia and $T = 1100$ F, based on the data in Table 4.10, using a least squares quadratic bivariate

polynomial. Let the variables x , y , and Z in Eq. (4.170) correspond to P , T , and h , respectively. Evaluating the summations and substituting into Eq. (4.170) gives

$$\left[\begin{array}{cccccc} 9 \text{ E0} & 10.800 \text{ E3} & 9.000 \text{ E3} & 12.975 \text{ E6} & 9.240 \text{ E6} & 10.800 \text{ E6} \\ 10.800 \text{ E3} & 12.975 \text{ E6} & 10.800 \text{ E6} & 15.606 \text{ E9} & 11.088 \text{ E9} & 12.975 \text{ E9} \\ 9.000 \text{ E3} & 10.800 \text{ E6} & 9.240 \text{ E6} & 12.975 \text{ E9} & 9.720 \text{ E9} & 11.088 \text{ E9} \\ 12.975 \text{ E6} & 15.606 \text{ E9} & 12.975 \text{ E9} & 18.792 \text{ E12} & 13.321 \text{ E12} & 15.606 \text{ E12} \\ 9.240 \text{ E6} & 11.088 \text{ E9} & 9.720 \text{ E9} & 13.321 \text{ E12} & 10.450 \text{ E12} & 11.664 \text{ E12} \\ 10.800 \text{ E6} & 12.975 \text{ E9} & 11.088 \text{ E9} & 15.606 \text{ E12} & 11.664 \text{ E12} & 13.321 \text{ E12} \end{array} \right] \left[\begin{array}{l} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \\ \mathbf{e} \\ \mathbf{f} \end{array} \right] = \left[\begin{array}{l} 13.4703 \text{ E3} \\ 16.1638 \text{ E6} \\ 13.6118 \text{ E6} \\ 19.4185 \text{ E9} \\ 14.1122 \text{ E9} \\ 16.3337 \text{ E9} \end{array} \right] \quad (4.171)$$

Due to the large magnitudes of the coefficients, they are expressed in exponential format (i.e., $x.\text{xxx } En = x.\text{xxx} \times 10^n$). Each row in Eq. (4.171) should be normalized by the exponential term in the first coefficient of each row. Solving the normalized Eq. (4.171) by Gauss elimination yields

$$\begin{aligned} h(T, P) = & 914.033 - 0.0205000P + 0.645500T - 0.000040000P^2 \\ & - 0.000077500T^2 + 0.000082500PT \end{aligned} \quad (4.172)$$

Evaluating Eq. (4.172) yields $h(1100.0, 1225.0) = 1556.3$ Btu/lbm. The error is $\text{Error} = 1556.3 - 1556.0 = 0.3$ Btu/lbm. which is smaller than the error incurred in Example 4.12 obtained by direct multivariate linear interpolation.

Equation (4.170) can be written as the matrix equation

$$\mathbf{Ac} = \mathbf{b} \quad (4.173)$$

where \mathbf{A} is the 6×6 coefficient matrix, \mathbf{c} is the 6×1 column vector of polynomial coefficients (i.e., a to f), and \mathbf{b} is the 6×1 column vector of nonhomogeneous terms. The solution to Eq. (4.173) is

$$\mathbf{c} = \mathbf{A}^{-1}\mathbf{b} \quad (4.174)$$

where \mathbf{A}^{-1} is the inverse of \mathbf{A} . In general, solving Eq. (4.173) for \mathbf{c} by Gauss elimination is more efficient than calculating \mathbf{A}^{-1} . However, for equally spaced data (i.e. $\Delta x = \text{constant}$ and $\Delta y = \text{constant}$) in a large table, a considerable simplification can be achieved. This is accomplished by locally transforming the independent variables so that $x = y = 0$ at the center point of each subset of data. In this case, \mathbf{A} is constant for the entire table, so \mathbf{A}^{-1} can be determined once for the entire table, and Eq. (4.174) can be used to calculate the coefficients a to f at any point in the table very efficiently. Only the nonhomogeneous vector \mathbf{b} changes from point to point and must be recalculated.

4.10.5. Nonlinear Functions

One advantage of using polynomials for least squares approximation is that the normal equations are linear, so it is straightforward to solve for the coefficients of the polynomials. In some engineering and science problems, however, the underlying physics suggests forms of approximating functions other than polynomials. Examples of several such functions are presented in this section.

Many physical processes are governed by a nonlinear equation of the form

$$y = ax^b \quad (4.175)$$

Taking the natural logarithm of Eq. (4.175) gives

$$\ln(y) = \ln(a) + b \ln(x) \quad (4.176)$$

Let $y' = \ln(y)$, $a' = \ln(a)$, and $x' = \ln(x)$. Equation (4.176) becomes

$$y' = a' + bx' \quad (4.177)$$

which is a linear polynomial. Equation (4.177) can be used as a least squares approximation by applying the results developed in Section 4.10.2.

Another functional form that occurs frequently in physical processes is

$$y = ae^{bx} \quad (4.178)$$

Taking the natural logarithm of Eq. (4.178) gives

$$\ln(y) = \ln(a) + bx \quad (4.179)$$

which can be written as the linear polynomial

$$y' = a' + bx \quad (4.180)$$

Equations (4.175) and (4.178) are examples of nonlinear functions that can be manipulated into a linear form. Some nonlinear functions cannot be so manipulated. For example, consider the nonlinear function

$$y = \frac{a}{1 + bx} \quad (4.181)$$

For a given set of data, (x_i, Y_i) ($i = 1, 2, \dots, N$), the sum of the squares of the deviations is

$$S(a, b) = \sum \left(Y_i - \frac{a}{1 + bx_i} \right)^2 \quad (4.182)$$

The function $S(a, b)$ is a minimum when

$$\frac{\partial S}{\partial a} = \sum 2 \left(Y_i - \frac{a}{1 + bx_i} \right) \left(-\frac{1}{1 + bx_i} \right) = 0 \quad (4.183a)$$

$$\frac{\partial S}{\partial b} = \sum 2 \left(Y_i - \frac{a}{1 + bx_i} \right) \left(\frac{x_i}{(1 + bx_i)^2} \right) = 0 \quad (4.183b)$$

Equation (4.183) comprises a pair of nonlinear equations for determining the coefficients a and b . They can be solved by methods for solving systems of nonlinear equations, for example, by Newton's method, as discussed in Section 3.7.

4.11 PROGRAMS

Five FORTRAN subroutines for fitting approximating polynomials are presented in this section:

1. Direct fit polynomials
2. Lagrange polynomials
3. Divided difference polynomials
4. Newton forward-difference polynomials
5. Least squares polynomials

All of the subroutines are written for a *quadratic* polynomial, except the *linear* least squares polynomial. Higher-degree polynomials can be constructed by following the patterns of the quadratic polynomials. The variable *ndeg*, which specifies the degree of the polynomials, is thus not needed in the present programs. It is included, however, to facilitate the addition of other-degree polynomials to the subroutines, in which case the degree of the polynomial to be evaluated must be specified.

The basic computational algorithms are presented as completely self-contained subroutines suitable for use in other programs. FORTRAN subroutines *direct*, *lagrange*, *divdiff* and *newtonfd* are presented in Sections 4.11.1 to 4.11.4 for implementing these procedures. A common *program main* defines the data sets and prints them, calls one of the subroutines to implement the solution, and prints the solution. The complete *program main* is presented with *subroutine direct*. For the other three subroutines, only the changed statements are included. *Program main* for *subroutine lesq* is completely self-contained.

4.11.1. Direct Fit Polynomials

The direct fit polynomial is specified by Eq. (4.34):

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \quad (4.184)$$

A FORTRAN subroutine, *subroutine direct*, for implementing a *quadratic* direct fit polynomial is presented below. *Program main* (Program 4.1) defines the data set and prints it, calls *subroutine direct* to implement the solution, and prints the solution. *Subroutine gauss* is taken from Section 1.8.1.

Program 4.1. Direct fit polynomial program.

```

program main
c      main program to illustrate polynomial fitting subroutines
c      ndim array dimension, ndim = 6 in this example
c      ndeg degree of the polynomial, ndeg = 2 in this example
c      n      number of data points and polynomial coefficients
c      xp     value of x at which the polynomial is evaluated
c      x      values of the independent variable, x(i)
c      f      values of the dependent variable, f(i)
c      fexp   interpolated value, f(xp)
c      c      coefficients of the direct fit polynomial, c(i)
dimension x(6),f(6),a(6,6),b(6),c(6)
data ndim,ndeg,n,xp / 6, 2, 3, 3.44 /
data (x(i),i=1,3) / 3.35, 3.40, 3.50 /
data (f(i),i=1,3) / 0.298507, 0.294118, 0.285714 /

```

```

write (6,1000)
do i=1,n
    write (6,1010) i,x(i),f(i)
end do
call direct (ndim,ndeg,n,x,f,xp,fxp,a,b,c)
write (6,1020) xp,fxp
stop
1000 format (' Quadratic direct fit polynomial'// '' x and f'// ')
1010 format (i4,2f12.6)
1020 format (' // f('' ,f6.3,'') = ',f12.6)
end

c subroutine direct (ndim,ndeg,n,x,f,xp,fxp,a,b,c)
c quadratic direct fit polynomial
dimension x(ndim),f(ndim),a(ndim,ndim),b(ndim),c(ndim)
do i=1,n
    a(i,1)=1.0
    a(i,2)=x(i)
    a(i,3)=x(i)**2
    b(i)=f(i)
end do
call gauss (ndim,n,a,b,c)
write (6,1000) (c(i),i=1,n)
fxp=c(1)+c(2)*xp+c(3)*xp**2
return
1000 format (' // f(x) = ',e13.7,' + ',e13.7,' x + ',e13.7,' x**2')
end

c subroutine gauss (ndim,n,a,b,x)
c implements simple Gauss elimination
end

```

The data set used to illustrate *subroutine direct* is taken from Example 4.2. The output generated by the direct fit polynomial program is presented below.

Output 4.1. Solution by a quadratic direct fit polynomial.

Quadratic direct fit polynomial

x and f

1	3.350000	0.298507
2	3.400000	0.294118
3	3.500000	0.285714

*f(x) = 0.8765607E+00 + -.2560800E+00 x + 0.2493333E-01 x**2*

f(3.440) = 0.290697

4.11.2. Lagrange Polynomials

The equation for the general Lagrange polynomial is given by Eq. (4.46). The quadratic Lagrange polynomial is given by Eq. (4.45):

$$P_2(x) = \frac{(x-b)(x-c)}{(a-b)(a-c)}f(a) + \frac{(x-a)(x-c)}{(b-a)(b-c)}f(b) + \frac{(x-a)(x-b)}{(c-a)(c-b)}f(c) \quad (4.185)$$

A FORTRAN subroutine, *subroutine lagrange*, for implementing the *quadratic* Lagrange polynomial is presented below. *Program main* (Program 4.2) defines the data set and prints it, calls *subroutine lagrange* to implement the solution, and prints the solution. It shows only the statements which are different from the statements in Program 4.1.

Program 4.2. Quadratic Lagrange polynomial program.

```

program main
c      main program to illustrate polynomial fitting subroutines
dimension x(6),f(6)
call lagrange (ndim,n,x,f,xp,fpx)
1000 format (' Quadratic Lagrange polynomial'// ' ' x and f'/' ')
end

subroutine lagrange (ndim,n,x,f,xp,fpx)
c      quadratic Lagrange polynomial
dimension x(ndim),f(ndim)
fp1=(xp-x(2))*(xp-x(3))/(x(1)-x(2))/(x(1)-x(3))*f(1)
fp2=(xp-x(1))*(xp-x(3))/(x(2)-x(1))/(x(2)-x(3))*f(2)
fp3=(xp-x(1))*(xp-x(2))/(x(3)-x(1))/(x(3)-x(2))*f(3)
fpx=fp1+fp2+fp3
return
end

```

The data set used to illustrate *subroutine lagrange* is taken from Example 4.3. The output generated by the Lagrange polynomial program is presented in Output 4.2.

Output 4.2. Solution by a quadratic Lagrange polynomial.

Quadratic Lagrange polynomial

x and *f*

1	3.350000	0.298507
2	3.400000	0.294118
3	3.500000	0.285714

f(3.440) = 0.290697

4.11.3. Divided Difference Polynomial

The general formula for the divided difference polynomial is given by Eq. (4.65):

$$P_n(x) = f_i^{(0)} + (x - x_0)f_i^{(1)} + (x - x_0)(x - x_1)f_i^{(2)} + \cdots + (x - x_0)(x - x_1)\cdots(x - x_{n-1})f_i^{(n)} \quad (4.186)$$

A FORTRAN subroutine, *subroutine divdiff*, for implementing a quadratic divided difference polynomial is presented below. *Program main* (Program 4.3) defines the data set and prints it, calls *subroutine divdiff* to implement the solution, and prints the solution. It shows only the statements which are different from the statements in Program 4.1.

Program 4.3. Quadratic divided difference polynomial program.

```

program main
c   main program to illustrate polynomial fitting subroutines
dimension x(6),f(6)
call divdiff (ndim,n,x,f,xp,fpx)
1000 format (' Quadratic divided diff. poly.'// ' x and f'// ' ')
      end

      subroutine divdiff (ndim,n,x,f,xp,fpx)
c   quadratic divided difference polynomial
dimension x(ndim),f(ndim)
f11=(f(2)-f(1))/(x(2)-x(1))
f21=(f(3)-f(2))/(x(3)-x(2))
f12=(f21-f11)/(x(3)-x(1))
fpx=f(1)+(xp-x(1))*f11+(xp-x(2))*(xp-x(2))*f12
      return
      end

```

The data set used to illustrate *subroutine divdiff* is taken from Example 4.6. The output generated by the divided difference polynomial program is presented in Output 4.3.

Output 4.3. Solution by a quadratic divided difference polynomial.

Quadratic divided diff. polynomial

x and f

1	3.350000	0.298507
2	3.400000	0.294118
3	3.500000	0.285714

f(3.440) = 0.290697

4.11.4. Newton Forward-Difference Polynomial

The general formula for the Newton forward-difference polynomial is given by Eq. (4.88):

$$\begin{aligned} P_n(x) = f_0 + s \Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!} \Delta^3 f_0 \\ + \cdots + \frac{s(s-1)(s-2) \cdots [s-(n-1)]}{n!} \Delta^n f_0 \end{aligned} \quad (4.187)$$

A FORTRAN subroutine, *subroutine newtonfd*, for implementing a quadratic Newton forward difference polynomial is presented below. *Program main* (Program 4.4) defines the data set and prints it, calls *subroutine newtonfd* to implement the solution, and prints the solution. It shows only the statements which are different from the statements in Program 4.1.

Program 4.4. Quadratic Newton forward-difference polynomial program.

```

program main
c main program to illustrate polynomial fitting subroutines
dimension x(6),f(6)
data (x(i),i=1,3) / 3.40, 3.50, 3.60 /
data (f(i),i=1,3) / 0.294118, 0.285714, 0.277778 /
call newtonfd (ndim,n,x,f,xp,fxp)
1000 format (' Quadratic Newton FD polynomial'/' '' x and f'/' ')
end

subroutine newtonfd (ndim,n,x,f,xp,fxp)
c quadratic Newton forward-difference polynomial
dimension x(ndim),f(ndim)
d11=f(2)-f(1)
d12=f(3)-f(2)
d21=d12-d11
s=(xp-x(1))/(x(2)-x(1))
fxp=f(1)+s*d11+s*(s-1.0)/2.0*d21
return
end

```

The data set used to illustrate *subroutine newtonfd* is taken from Example 4.8. The output generated by the Newton forward-difference polynomial program is presented in Output 4.4.

Output 4.4. Solution by quadratic Newton forward-difference polynomial.

Quadratic Newton FD polynomial

x and f

1	3.400000	0.294118
2	3.500000	0.285714
3	3.600000	0.277780

f(3.440) = 0.290700

4.11.5. Least Squares Polynomial

The normal equations for a general least squares polynomial are given by Eq. (4.158). The normal equations for a linear least squares polynomial are given by Eq. (4.149):

$$aN + b \sum_{i=1}^N x_i = \sum_{i=1}^N Y_i \quad (4.188a)$$

$$a \sum_{i=1}^N x_i + b \sum_{i=1}^N x_i^2 = \sum_{i=1}^N x_i Y_i \quad (4.188b)$$

A FORTRAN subroutine, *subroutine lesq*, for implementing a *linear* least squares polynomial is presented below. *Program main* (Program 4.5) defines the data set and prints it, calls *subroutine lesq* to implement the solution, and prints the solution.

Program 4.5. Linear least squares polynomial program.

```

program main
c      main program to illustrate subroutine lesq
c      ndim  array dimension, ndim = 10 in this example
c      ndeg  degree of the polynomial
c      n      number of data points and polynomial coefficients
c      x      values of the independent variable, x(i)
c      f      values of the dependent variable, f(i)
c      c      coefficients of the least squares polynomials, c(i)
dimension x(10),f(10),a(10,10),b(10),c(10)
data ndim,ndeg,n / 10, 1, 8 /
data (x(i), i=1,8) / 300.0, 400.0, 500.0, 600.0, 700.0,
1 800.0, 900.0, 1000.0 /
data (f(i), i=1,8) / 1.0045, 1.0134, 1.0296, 1.0507, 1.0743,
1 1.0984, 1.1212, 1.1410 /
write (6,1000)
call lesq (ndim,ndeg,n,x,f,a,b,c)
write (6,1010)
do i=1,n
    fi=c(1)+c(2)*x(i)
    error=100.0*(f(i)-fi)
    write (6,1020) i,x(i),f(i),fi,error
end do
stop
1000 format (' Linear least squares polynomial')
1010 format ('   ',i',5x,'x',9x,'f',9x,'fi',7x,'error'//  ')
1020 format (i4,f10.3,2f10.4,f9.2)
end

subroutine lesq (ndim,ndeg,n,x,f,a,b,c)
c      linear least squares polynomial
dimension x(ndim),f(ndim),a(ndim,ndim),b(ndim),c(ndim)
sumx=0.0
sumxx=0.0
sumf=0.0
sumxf=0.0

```

```

do i=1,n
    sumx=sumx+x(i)
    sumxx=sumxx+x(i)**2
    sumf=sumf+f(i)
    sumxf=sumxf+x(i)*f(i)
end do
a(1,1)=float(n)
a(1,2)=sumx
a(2,1)=sumx
a(2,2)=sumxx
b(1)=sumf
b(2)=sumxf
call gauss (ndim,ndeg+1,a,b,c)
write (6,1000) (c(i),i=1,ndeg+1)
return
1000 format (' // f(x) = ',e13.7,' + ',e13.7,' x')
end

subroutine gauss (ndim,n,a,b,x)
c implements simple gauss elimination
end

```

The data set used to illustrate *subroutine lesq* is taken from Example 4.14. The output generated by the linear least squares polynomial program is presented in Output 4.5.

Output 4.5. Solution by a linear least squares polynomial.

Linear least squares polynomial

$$f(x) = 0.9331940E+00 + 0.2052976E-03 x$$

i	x	f	fi	error
1	300.000	1.0045	0.9948	0.97
2	400.000	1.0134	1.0153	-0.19
3	500.000	1.0296	1.0358	-0.62
4	600.000	1.0507	1.0564	-0.57
5	700.000	1.0743	1.0769	-0.26
6	800.000	1.0984	1.0974	0.10
7	900.000	1.1212	1.1180	0.32
8	1000.000	1.1410	1.1385	0.25

4.11.6. Packages for Polynomial Approximation

Numerous libraries and software packages are available for polynomial approximation and interpolation. Many workstations and mainframe computers have such libraries attached to their operating systems.

Many commercial software packages contain routines for fitting approximating polynomials and interpolation. Some of the more prominent packages are Matlab and Mathcad. More sophisticated packages, such as IMSL, Mathematica, Macsyma, and

Maple, also contain routines for fitting approximating polynomials and interpolation. Finally, the book *Numerical Recipes*, Press et al. (1989) contains numerous subroutines for fitting approximating polynomials and interpolation.

4.12 SUMMARY

Procedures for developing approximating polynomials for discrete data are presented in this chapter. For small sets of smooth data, exact fits are desirable. The direct fit polynomial, the Lagrange polynomial, and the divided difference polynomial work well for nonequally spaced data. For equally spaced data, polynomials based on differences are recommended. The Newton forward-difference and backward-difference polynomials are simple to fit and evaluate. These two polynomials are used extensively in Chapters 5 and 6 to develop procedures for numerical differentiation and numerical integration, respectively. Procedures are discussed for inverse interpolation and multivariate approximation.

Procedures for developing least squares approximations for discrete data also are presented in this chapter. Least squares approximations are useful for large sets of data and sets of rough data. Least squares polynomial approximation is straightforward, for both one independent variable and more than one independent variable. The least squares normal equations corresponding to polynomial approximating functions are linear, which leads to very efficient solution procedures. For nonlinear approximating functions, the least squares normal equations are nonlinear, which leads to complicated solution procedures. Least squares polynomial approximation is a straightforward, simple, and accurate procedure for obtaining approximating functions for large sets of data or sets of rough data.

After studying Chapter 4, you should be able to:

1. Discuss the general features of functional approximation
2. List the uses of functional approximation
3. List several types of approximating functions
4. List the required properties of an approximating function
5. Explain why polynomials are good approximating functions
6. List the types of discrete data which influence the type of fit required
7. Describe the difference between exact fits and approximate fits
8. Explain the significance of the Weirstrass approximation theorem
9. Explain the significance of the uniqueness theorem for polynomials
10. List the Taylor series and the Taylor polynomial
11. Discuss the error term of a polynomial
12. Evaluate, differentiate, and integrate a polynomial
13. Explain and apply the nested multiplication algorithm
14. State and apply the division algorithm, the remainder theorems, and the factor theorem
15. Explain and apply the synthetic division algorithm
16. Explain and implement polynomial deflation
17. Discuss the advantages and disadvantages of direct fit polynomials
18. Fit a direct fit polynomial of any degree to a set of tabular data
19. Explain the concept underlying Lagrange polynomials
20. Discuss the advantages and disadvantages of Lagrange polynomials
21. Fit a Lagrange polynomial of any degree to a set of tabular data

22. Discuss the advantages and disadvantages of Neville's algorithm
23. Apply Neville's algorithm
24. Define a divided difference and construct a divided difference table
25. Discuss the advantages and disadvantages of divided difference polynomials
26. Construct a divided difference polynomial
27. Define a difference and construct a difference table
28. Discuss the effects of round-off on a difference table
29. Discuss the advantages and disadvantages of difference polynomials.
30. Apply the Newton forward-difference polynomial
31. Apply the Newton backward-difference polynomial
32. Discuss the advantages of centered difference polynomials
33. Discuss and apply inverse interpolation
34. Describe approaches for multivariate approximation
35. Apply successive univariate polynomial approximation
36. Apply direct multivariate polynomial approximation
37. Describe the concepts underlying cubic splines
38. List the advantages and disadvantages of cubic splines
39. Apply cubic splines to a set of tabular data
40. Discuss the concepts underlying least squares approximation
41. List the advantages and disadvantages of least squares approximation
42. Derive the normal equations for a least squares polynomial of any degree
43. Apply least squares polynomials to fit a set of tabular data
44. Derive and solve the normal equations for a least squares approximation of a nonlinear function
45. Apply the computer programs presented in Section 4.11 to fit polynomials and to interpolate
46. Develop computer programs for applications not implemented in Section 4.11.

EXERCISE PROBLEMS

4.2 Properties of Polynomials

1. For the polynomial $P_3(x) = x^3 - 9x^2 + 26x - 24$, calculate (a) $P_3(1.5)$ by nested multiplication, (b) $P'_3(1.5)$ by constructing $Q_2(x)$ and evaluating $Q_2(1.5)$ by nested multiplication, and (c) the deflated polynomial $Q_2(x)$ obtained by removing the factor $(x - 2)$.
2. Work Problem 1 for $x = 2.5$ and remove the factor $(x - 3)$.
3. Work Problem 1 for $x = 3.5$ and remove the factor $(x - 4)$.
4. For the polynomial $P_4(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$, calculate (a) $P_4(1.5)$ by nested multiplication, (b) $P'_4(1.5)$ by constructing $Q_3(x)$ and evaluating $Q_3(1.5)$ by nested multiplication, and (c) the deflated polynomial $Q_3(x)$ obtained by removing the factor $(x - 1)$.
5. Work Problem 4 for $x = 2.5$ and remove the factor $(x - 2)$.
6. Work Problem 4 for $x = 3.5$ and remove the factor $(x - 3)$.
7. Work Problem 4 for $x = 4.5$ and remove the factor $(x - 4)$.
8. For the polynomial $P_5(x) = x^5 - 20x^4 + 155x^3 - 580x^2 + 1044x - 720$, calculate (a) $P_5(1.5)$ by nested multiplication, (b) $P'_5(1.5)$ by constructing $Q_4(x)$

- and evaluating $Q_4(1.5)$ by nested multiplication, and (c) the deflated polynomial $Q_4(x)$ obtained by removing the factor $(x - 2)$.
9. Work Problem 8 for $x = 2.5$ and remove the factor $(x - 3)$.
 10. Work Problem 8 for $x = 3.5$ and remove the factor $(x - 4)$.
 11. Work Problem 8 for $x = 4.5$ and remove the factor $(x - 5)$.
 12. Work Problem 8 for $x = 5.5$ and remove the factor $(x - 6)$.

4.3 Direct Fit Polynomials

Table 1 is for the function $f(x) = 2/x + x^2$. This table is considered in several of the problems which follow.

Table 1. Tabular Data.

x	$f(x)$	x	$f(x)$	x	$f(x)$
0.4	5.1600	1.4	3.3886	2.2	5.7491
0.6	3.6933	1.6	3.8100	2.4	6.5933
0.8	3.1400	1.8	4.3511	2.6	7.5292
1.0	3.0000	2.0	5.0000	2.8	8.5543
1.2	3.1067				

13. The order n of an approximating polynomial specifies the rate at which the error of the polynomial approximation approaches zero as the increment in the tabular data approaches zero, that is, $\text{Error} = C \Delta x^n$. Estimate the order of a linear direct fit polynomial by calculating $f(2.0)$ for the data in Table 1 using $x = 1.6$ and 2.4 (i.e., $\Delta x = 0.8$), and $x = 1.8$ and 2.2 (i.e., $\Delta x = 0.4$), calculating the errors, and calculating the ratio of the errors.
14. Consider the tabular set of generic data shown in Table 2: Determine the direct fit quadratic polynomials for the tabular set of generic data for (a) $\Delta x_- = \Delta x_+ = \Delta x$ and (b) $\Delta x_- \neq \Delta x_+$.

Table 2. Generic Data

x	$f(x)$
$x_{i-1} = -\Delta x_-$	f_{i-1}
$x_i = 0$	f_i
$x_{i+1} = +\Delta x_+$	f_{i+1}

15. The formal order of a direct fit polynomial $P_n(x)$ can be determined by expressing all function values in the polynomial (i.e., f_{i-1}, f_{i+1} , etc.) in terms of a Taylor series at the base point and comparing that result to the Taylor series for $f(x)$ at the base point. For the direct fit polynomials developed in Problem 14, show that the order is $O(\Delta x^3)$ for part (a) and $O(\Delta x_-^2) + O(\Delta x_+^2)$ for part (b).
16. Consider the data in the range $0.4 \leq x \leq 1.2$ in Table 1. Using direct fit polynomials, calculate (a) $P_2(0.9)$ using the first three points, (b) $P_2(0.9)$ using the last three points, (c) $P_3(0.9)$ using the first four points, (d) $P_3(0.9)$ using the last four points, and (e) $P_4(0.9)$ using all five data points.
17. Work Problem 16 for the range $1.2 \leq x \leq 2.0$ for $x = 1.5$.

18. Work Problem 16 for the range $2.0 \leq x \leq 2.8$ for $x = 2.5$.
19. The constant pressure specific heat C_p and enthalpy h of low pressure air are tabulated in Table 3. Using direct fit polynomials with the base point as close to the specified value of T as possible, calculate (a) $C_p(1120)$ using two points, (b) $C_p(1120)$ using three points, (c) $C_p(1480)$ using two points, and (d) $C_p(1480)$ using three points.

Table 3. Properties of Air

T, K	$C_p, \text{ kJ/kg-K}$	$h, \text{ kJ/kg}$	T, K	$C_p, \text{ kJ/kg-K}$	$h, \text{ kJ/kg}$
1000	1.1410	1047.248	1400	1.1982	1515.792
1100	1.1573	1162.174	1500	1.2095	1636.188
1200	1.1722	1278.663	1600	1.2197	1757.657
1300	1.1858	1396.578			

20. Work Problem 19 for $h(T)$ instead of $C_p(T)$.

4.4 Lagrange Polynomials

21. Work Problem 16 using Lagrange polynomials.
22. Work Problem 17 using Lagrange polynomials.
23. Work Problem 18 using Lagrange polynomials.
24. Work Problem 19 using Lagrange polynomials.
25. Work Problem 20 using Lagrange polynomials.
26. Work Problem 16 using Neville's algorithm.
27. Work Problem 17 using Neville's algorithm.
28. Work Problem 18 using Neville's algorithm.
29. Work Problem 19 using Neville's algorithm.
30. Work Problem 20 using Neville's algorithm.

4.5 Divided Difference Tables and Divided Difference Polynomials

Divided Difference Tables

31. Construct a six-place divided difference table for the function $f(x) = x^3 - 9x^2 + 26x - 24$ in the range $1.1 \leq x \leq 1.9$ for $x = 1.10, 1.15, 1.20, 1.30, 1.35, 1.45, 1.6, 1.75$, and 1.90 .
32. Construct a divided difference table for the data in Table 1.
33. Construct a divided difference table for $C_p(T)$ for the data presented in Table 3.
34. Construct a divided difference table for $h(T)$ for the data presented in Table 3.

Divided Difference Polynomials

35. From the divided difference table constructed in Problem 31, interpolate for $f(1.44)$ using: (a) two points, (b) three points, and (c) four points. In each case use the closest points to $x = 1.44$.
36. Using the divided difference table constructed in Problem 32, evaluate $f(0.9)$ using: (a) two points, (b) three points, and (c) four points. In each case use the closest points to $x = 0.9$.

37. From the divided difference table constructed in Problem 33, calculate $C_p(1120)$ using: (a) two points, (b) three points, and (c) four points. In each case use the closest points to $T = 1120$ K.
38. From the divided difference table constructed in Problem 34, calculate $h(1120)$ using: (a) two points, (b) three points, and (c) four points. In each case use the closest points to $T = 1120$ K.

4.6 Difference Tables and Difference Polynomials

Difference Tables

39. Construct a six-place difference table for the function $f(x) = x^3 - 9x^2 + 26x - 24$ in the range $1.1 \leq x \leq 1.9$ for $\Delta x = 0.1$. Discuss the results. Analyze the effects of round off.
40. Construct a difference table for the data in Table 1. Discuss the results. Analyze the effects of round off. Comment on the degree of polynomial required to approximate these data at the beginning, middle, and end of the table.
41. Construct a difference table for $C_p(T)$ for the data presented in Table 3. Discuss the results. Analyze the effects of round-off. What degree of polynomial is required to approximate this set of data?
42. Work Problem 41 for $h(T)$.

The Newton Forward-Difference Polynomial

43. Work Problem 16 using Newton forward-difference polynomials.
44. Work Problem 17 using Newton forward-difference polynomials.
45. Work Problem 18 using Newton forward-difference polynomials.
46. Work Problem 19 using Newton forward-difference polynomials.
47. Work Problem 20 using Newton forward-difference polynomials.

The Newton Backward-Difference Polynomial

48. Work Problem 16 using Newton backward-difference polynomials.
49. Work Problem 17 using Newton backward-difference polynomials.
50. Work Problem 18 using Newton backward-difference polynomials.
51. Work Problem 19 using Newton backward-difference polynomials.
52. Work Problem 20 using Newton backward-difference polynomials.
53. For the data in Table 3 in the temperature range $1200 \leq T \leq 1400$, construct a quadratic direct fit polynomial, a quadratic Newton forward-difference polynomial, and a quadratic Newton backward-difference polynomial for $C_p(T)$. Rewrite both Newton polynomials in the form $C_p(T) = a + bT + CT^2$, and show that the three polynomials are identical.
54. Work Problem 53 including a quadratic Lagrange polynomial.

Other Difference Polynomials

55. Work Problem 16 using the Stirling centered-difference polynomial.
56. Work Problem 17 using the Stirling centered-difference polynomial.
57. Work Problem 18 using the Stirling centered-difference polynomial.
58. Work Problem 19 using the Stirling centered-difference polynomial.
59. Work Problem 20 using the Stirling centered-difference polynomial.

60. Work Problem 16 using the Bessel centered-difference polynomial.
61. Work Problem 17 using the Bessel centered-difference polynomial.
62. Work Problem 18 using the Bessel centered-difference polynomial.
63. Work Problem 19 using the Bessel centered-difference polynomial.
64. Work Problem 20 using the Bessel centered-difference polynomial.

4.8 Multivariate Interpolation

65. The specific volume v (m^3/kg) of steam, corresponding to the van der Waal equation of state (see Problem 3.69), as a function of pressure P (kN/m^2) and temperature T (K) in the neighborhood of $P = 10,000 \text{ kN/m}^2$ and $T = 800 \text{ K}$, is tabulated in Table 4.

Table 4. Specific Volume of Steam

$P, \text{kN/m}^2$	T, K		
	700	800	900
9,000	0.031980	0.037948	0.043675
10,000	0.028345	0.033827	0.039053
11,000	0.025360	0.030452	0.035270

Use successive quadratic univariate interpolation to calculate $v(9500, 750)$. The exact value from the steam tables is $v = 0.032965 \text{ m}^3/\text{kg}$.

66. Work Problem 65 for $v(9500, 850)$. The exact value is $0.038534 \text{ m}^3/\text{kg}$.
67. Work Problem 65 for $v(10500, 750)$. The exact value is $0.029466 \text{ m}^3/\text{kg}$.
68. Work Problem 65 for $v(10500, 850)$. The exact value is $0.034590 \text{ m}^3/\text{kg}$.
69. Solve Problem 65 by direct linear bivariate interpolation for $v(9500, 750)$:

$$v = a + bT + cP + dPT$$

70. Work Problem 69 for $v(9500, 850)$.
71. Work Problem 69 for $v(10500, 750)$.
72. Work Problem 69 for $v(10500, 850)$.
73. Solve Problem 65 by direct quadratic bivariate interpolation for $v(9500, 750)$:

$$v = a + bT + cP + dPT + eT^2 + fP^2$$

74. Work Problem 73 for $v(9500, 850)$.
75. Work Problem 73 for $v(10500, 750)$.
76. Work Problem 73 for $v(10500, 850)$.

4.10 Least Squares Approximation

77. Consider the data for the specific heat C_p of air, presented in Table 3 for the range $1,000 \leq T \leq 1,400$. Find the best straight line approximation to this set of data. Compute the deviations at each data point.
78. Work Problem 77 using every other data point. Compare the results with the results of Problem 77.
79. Work Problem 77 for a quadratic polynomial. Compare the results with the results of Problem 77.

80. Consider the data for the specific volume of steam, $v = v(P, T)$, given in Table 4. Develop a least squares linear bivariate polynomial for the set of data in the form

$$v = a + bT + cP + dPT$$

Compute the derivation at each data point. Calculate $v(9500, 750)$ and compare with the result from Problem 69.

81. Work Problem 80 for the least squares quadratic bivariate polynomial and compute the deviations.

$$v = a + bT + cP + dPT + eT^2 + fP^2$$

Compare the result with the result from Problem 73.

82. Fit the $C_p(T)$ data in Table 3 to a fourth-degree polynomial and compute the deviations.

$$C_p(T) = a + bT + cT^2 + dT^3 + eT^4$$

4.11 Programs

83. Implement the quadratic direct fit polynomial program presented in Section 4.11.1. Check out the program with the given data.
84. Solve any of Problems 16 to 20 with the program.
85. Modify the quadratic direct fit polynomial program to consider a linear direct fit polynomial. Solve any of Problems 16 to 20 with the modified program.
86. Modify the quadratic direct fit polynomial program to consider a cubic direct fit polynomial. Solve any of Problems 16 to 20 with the modified program.
87. Implement the quadratic Lagrange polynomial program presented in Section 4.11.2. Check out the program with the given data.
88. Solve any of Problems 16 to 20 with the program.
89. Modify the quadratic Lagrange polynomial program to consider a linear Lagrange polynomial. Solve any of Problems 16 to 20 with the modified program.
90. Modify the quadratic Lagrange polynomial program to consider a cubic Lagrange polynomial. Solve any of Problems 16 to 20 with the modified program.
91. Implement the quadratic divided difference polynomial program presented in Section 4.11.3. Check out the program with the given data.
92. Solve any of Problems 16 to 20 with the program.
93. Modify the quadratic divided difference polynomial program to consider a linear divided difference polynomial. Solve any of Problems 16 to 20 with the modified program.
94. Modify the quadratic divided difference polynomial program to consider a cubic divided difference polynomial. Solve any of Problems 16 to 20 with the modified program.
95. Implement the quadratic Newton forward-difference polynomial program presented in Section 4.11.4. Check out the program with the given data.
96. Solve any of Problems 16 to 20 with the program.
97. Modify the quadratic Newton forward-difference polynomial program to consider a linear Newton forward-difference polynomial. Solve any of Problems 16 to 20 with the modified program.

98. Modify the quadratic Newton forward-difference polynomial program to consider a cubic Newton forward-difference polynomial. Solve any of Problems 16 to 20 with the modified program A.
99. Implement the linear least squares polynomial program presented in Section 4.11.5. Check out the program with the given data.
100. Solve Problem 77 or 78 with the program.
101. Extend the linear least squares polynomial program to consider a quadratic least squares polynomial. Solve Problem 77 using the program.
102. Extend the linear least squares polynomial program to consider a fourth-degree least squares polynomial. Solve Problem 82 using the program.
103. Modify the linear least squares polynomial program to consider a linear bivariate least squares polynomial. Solve Problem 80 with the program.
104. Modify the linear least squares polynomial program to consider a quadratic bivariate least squares polynomial. Solve Problem 81 with the program.

APPLIED PROBLEMS

105. When an incompressible fluid flows steadily through a round pipe, the pressure drop ΔP due to friction is given by

$$\Delta P = -0.5f \rho V^2 (L/D)$$

where ρ is the fluid density, V is the velocity, L/D is the pipe length-to-diameter ratio, and f is the D'Arcy friction coefficient. For laminar flow, the friction coefficient f can be related to the Reynolds number, Re , by a relationship of the form

$$f = a Re^b$$

Use the measured data in Table 5 to determine a and b by a least squares fit.

Table 5. Friction Coefficient

Re	500	1000	1500	2000
f	0.0320	0.0160	0.0107	0.0080

106. Reaction rates for chemical reactions are usually expressed in the form

$$K = BT^\alpha \exp\left(\frac{-E}{RT}\right)$$

For a particular reaction, measured values of the forward and backward reaction rates K_f and K_b , respectively, are given by

Table 6. Reaction Rates

T, K	K_f	K_b
1000	7.5 E + 15	4.6 E + 07
2000	3.8 E + 15	5.9 E + 04
3000	2.5 E + 15	2.5 E + 08
4000	1.9 E + 15	1.4 E + 10
5000	1.5 E + 15	1.5 E + 11

- (a) Determine B and α for the backward reaction rate K_b for which $E/R = 0$.
(b) Determine B , α and E/R for the forward reaction rate K_f .
107. The data in Table 1 can be fit by the expression

$$f = \frac{a}{x} + bx^2$$

Develop a least squares procedure to determine a and b . Solve for a and b and compute the deviations.

5

Numerical Differentiation and Difference Formulas

- 5.1. Introduction
- 5.2. Unequally Spaced Data
- 5.3. Equally Spaced Data
- 5.4. Taylor Series Approach
- 5.5. Difference Formulas
- 5.6. Error Estimation and Extrapolation
- 5.7. Programs
- 5.8. Summary
Problems

Examples

- 5.1. Direct fit, Lagrange, and divided difference polynomials
- 5.2. Newton forward-difference polynomial, one-sided
- 5.3. Newton forward-difference polynomial, centered
- 5.4. Newton polynomial difference formulas
- 5.5. Taylor series difference formulas
- 5.6. Third-order nonsymmetrical difference formula for f_x
- 5.7. Error estimation and extrapolation

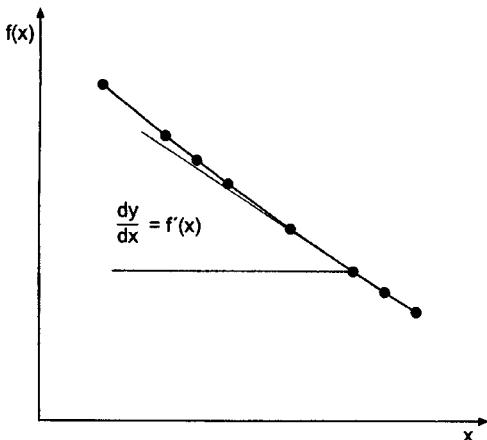
5.1 INTRODUCTION

Figure 5.1 presents a set of tabular data in the form of a set of $[x, f(x)]$ pairs. The function $f(x)$ is known only at discrete values of x . Interpolation within a set of discrete data is discussed in Chapter 4. *Differentiation* within a set of discrete data is presented in this chapter. The discrete data presented in Figure 5.1 are values of the function $f(x) = 1/x$, which are used as the example problem in this chapter.

The evaluation of a derivative is required in many problems in engineering and science:

$$\frac{d}{dx}(f(x)) = f'(x) = f_x(x)$$

(5.1)



x	$f(x)$
3.20	0.312500
3.30	0.303030
3.35	0.298507
3.40	0.294118
3.50	0.285714
3.60	0.277778
3.65	0.273973
3.70	0.270270

Figure 5.1 Differentiation of tabular data.

where the alternate notations $f'(x)$ and $f'_x(x)$ are used for the derivatives. The function $f(x)$, which is to be differentiated, may be a known function or a set of discrete data. In general, known functions can be differentiated exactly. Differentiation of discrete data, however, requires an approximate numerical procedure. The evaluation of derivatives by approximate numerical procedures is the subject of this chapter.

Numerical differentiation formulas can be developed by fitting approximating functions (e.g., polynomials) to a set of discrete data and differentiating the approximating function. Thus,

$$\frac{d}{dx}(f(x)) \cong \frac{d}{dx}(P_n(x)) \quad (5.2)$$

This process is illustrated in Figure 5.2. As illustrated in Figure 5.2, even though the approximating polynomial $P_n(x)$ passes through the discrete data points exactly, the derivative of the polynomial $P'_n(x)$ may not be a very accurate approximation of the derivative of the exact function $f(x)$ even at the known data points themselves. In general, numerical differentiation is an inherently inaccurate process.

To perform numerical differentiation, an approximating polynomial is fit to the discrete data, or a subset of the discrete data, and the approximating polynomial is

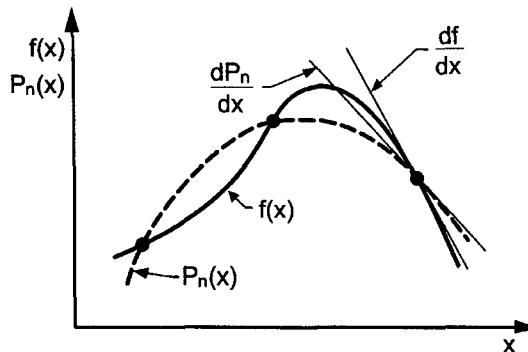


Figure 5.2 Numerical differentiation.

differentiated. The polynomial may be fit exactly to a set of discrete data by the methods presented in Sections 4.3 to 4.9, or approximately by a least squares fit as described in Section 4.10. In both cases, the degree of the approximating polynomial chosen to represent the discrete data is the only parameter under our control.

Several numerical differentiation procedures are presented in this chapter. Differentiation of direct fit polynomials, Lagrange polynomials, and divided difference polynomials can be applied to both unequally spaced data and equally spaced data. Differentiation formulas based on both Newton forward-difference polynomials and Newton backward-difference polynomials can be applied to equally spaced data. Numerical differentiation formulas can also be developed using Taylor series. This approach is quite useful for developing difference formulas for approximating exact derivatives in the numerical solution of differential equations.

The simple function

$$f(x) = \frac{1}{x} \quad (5.3)$$

which has the exact derivatives

$$\frac{d}{dx} \left(\frac{1}{x} \right) = f'(x) = -\frac{1}{x^2} \quad (5.4a)$$

$$\frac{d^2}{dx^2} \left(\frac{1}{x} \right) = f''(x) = \frac{2}{x^3} \quad (5.4b)$$

is considered in this chapter to illustrate numerical differentiation procedures. In particular, at $x = 3.5$:

$$f'(3.5) = -\frac{1}{(3.5)^2} = -0.081633\dots \quad (5.5a)$$

$$f''(3.5) = \frac{2}{(3.5)^3} = 0.046647\dots \quad (5.5b)$$

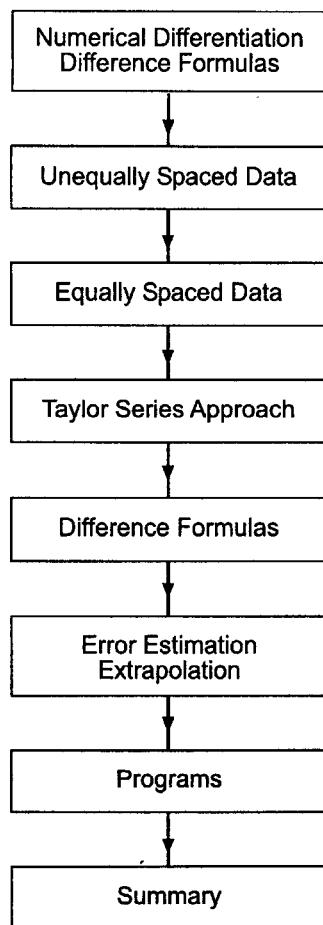


Figure 5.3 Organization of Chapter 5.

The organization of Chapter 5 is illustrated in Figure 5.3. Following the introductory discussion in this section, differentiation using direct fit polynomials, Lagrange polynomials, and divided difference polynomials as the approximating function is discussed. The development of differentiation formulas based on Newton difference polynomials is presented in Section 5.3. Section 5.4 develops difference formulas directly from the Taylor series. A table of difference formulas is presented in Section 5.5. A discussion of error estimation and extrapolation is presented in Section 5.6. Several programs for differentiating tabular data numerically are presented in Section 5.7. The chapter closes with a Summary which includes a list of what you should be able to do after studying Chapter 5.

5.2 UNEQUALLY SPACED DATA

Three straightforward numerical differentiation procedures that can be used for both unequally spaced data and equally spaced data are presented in this section:

1. Direct fit polynomials
2. Lagrange polynomials
3. Divided difference polynomials

5.2.1 Direct Fit Polynomials

A direct fit polynomial procedure is based on fitting the data directly by a polynomial and differentiating the polynomial. Recall the direct fit polynomial, Eq. (4.34):

$$P_n(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \quad (5.6)$$

where $P_n(x)$ is determined by one of the following methods:

1. Given $N = n + 1$ points, $[x_i, f(x_i)]$, determine the exact n th-degree polynomial that passes through the data points, as discussed in Section 4.3.
2. Given $N > n + 1$ points, $[x_i, f(x_i)]$, determine the least squares n th-degree polynomial that best fits the data points, as discussed in Section 4.10.3.

After the approximating polynomial has been fit, the derivatives are determined by differentiating the approximating polynomial. Thus,

$$f'(x) \cong P'_n(x) = a_1 + 2a_2x + 3a_3x^2 + \cdots \quad (5.7a)$$

$$f''(x) \cong P''_n(x) = 2a_2 + 6a_3x + \cdots \quad (5.7b)$$

Equations (5.7a) and (5.7b) are illustrated in Example 5.1.

5.2.2. Lagrange Polynomials

The second procedure that can be used for both unequally spaced data and equally spaced data is based on differentiating a Lagrange polynomial. For example, consider the second-degree Lagrange polynomial, Eq. (4.45):

$$P_2(x) = \frac{(x - b)(x - c)}{(a - b)(a - c)}f(a) + \frac{(x - a)(x - c)}{(b - a)(b - c)}f(b) + \frac{(x - a)(x - b)}{(c - a)(c - b)}f(c) \quad (5.8)$$

Differentiating Eq. (5.8) yields:

$$f'(x) \cong P'_2(x) = \frac{2x - (b + c)}{(a - b)(a - c)}f(a) + \frac{2x - (a + c)}{(b - a)(b - c)}f(b) + \frac{2x - (a + b)}{(c - a)(c - b)}f(c) \quad (5.9a)$$

Differentiating Eq. (5.9a) yields:

$$f''(x) \cong P''_2(x) = \frac{2f(a)}{(a - b)(a - c)} + \frac{2f(b)}{(b - a)(b - c)} + \frac{2f(c)}{(c - a)(c - b)} \quad (5.9b)$$

Equations (5.9a) and (5.9b) are illustrated in Example 5.1.

5.2.3. Divided Difference Polynomials

The third procedure that can be used for both unequally spaced data and equally spaced data is based on differentiating a divided difference polynomial, Eq. (4.65):

$$P_n(x) = f_i^{(0)} + (x - x_0)f_i^{(1)} + (x - x_0)(x - x_1)f_i^{(2)} + (x - x_0)(x - x_1)(x - x_2)f_i^{(3)} + \cdots \quad (5.10)$$

Differentiating Eq. (5.10) gives

$$\begin{aligned} f'(x) &\cong P'_n(x) = f_i^{(1)} + [2x - (x_0 + x_1)] f_i^{(2)} \\ &\quad + [3x^2 - 2(x_0 + x_1 + x_2)x + (x_0 x_1 + x_0 x_2 + x_1 x_2)] f_i^{(3)} + \dots \end{aligned} \quad (5.11a)$$

Differentiating Eq. (5.11a) gives

$$f''(x) \cong P''_n(x) = 2f_i^{(2)} + [6x - 2(x_0 + x_1 + x_2)] f_i^{(3)} + \dots \quad (5.11b)$$

Equations (5.10a) and (5.10b) are illustrated in Example 5.1.

Example 5.1. Direct fit, Lagrange, and divided difference polynomials.

Let's solve the example problem presented in Section 5.1 by the three procedures presented above. Consider the following three data points:

x	$f(x)$
3.4	0.294118
3.5	0.285714
3.6	0.277778

First, fit the quadratic polynomial, $P_2(x) = a_0 + a_1x + a_2x^2$, to the three data points:

$$0.294118 = a_0 + a_1(3.4) + a_2(3.4)^2 \quad (5.12a)$$

$$0.285714 = a_0 + a_1(3.5) + a_2(3.5)^2 \quad (5.12b)$$

$$0.277778 = a_0 + a_1(3.6) + a_2(3.6)^2 \quad (5.12c)$$

Solving for a_0 , a_1 , and a_2 by Gauss elimination gives $a_0 = 0.858314$, $a_1 = -0.245500$, and $a_2 = 0.023400$. Substituting these values into Eqs. (5.7a) and (5.7b) and evaluating at $x = 3.5$ yields the solution for the direct fit polynomial:

$$P'_2(3.5) = -0.245500 + (0.04680)(3.5) = -0.081700 \quad (5.12d)$$

$$P''_2(x) = 0.046800 \quad (5.12e)$$

Substituting the tabular values into Eqs. (5.9a) and (5.9b) and evaluating at $x = 3.5$ yields the solution for the Lagrange polynomial:

$$\begin{aligned} P'_2(3.5) &= \frac{2(3.5) - (3.5 + 3.6)}{(3.4 - 3.5)(3.4 - 3.6)} (0.294118) + \frac{2(3.5) - (3.4 + 3.6)}{(3.5 - 3.4)(3.5 - 3.6)} (0.285714) \\ &\quad + \frac{2(3.5) - (3.4 + 3.5)}{(3.6 - 3.4)(3.6 - 3.5)} (0.277778) = -0.081700 \end{aligned} \quad (5.13a)$$

$$\begin{aligned} P''_2(3.5) &= \frac{2(0.294118)}{(3.4 - 3.5)(3.4 - 3.6)} + \frac{2(0.285714)}{(3.5 - 3.4)(3.5 - 3.6)} + \frac{2(0.277778)}{(3.6 - 3.4)(3.6 - 3.5)} \\ &= 0.046800 \end{aligned} \quad (5.13b)$$

A divided difference table must be constructed for the tabular data to use the divided difference polynomial. Thus,

x_i	$f_i^{(0)}$	$f_i^{(1)}$	$f_i^{(2)}$
3.4	0.294118	-0.084040	
3.5	0.285714	-0.079360	0.023400
3.6	0.277778		

Substituting these values into Eqs. (5.11a) and (5.11b) yields the solution for the divided difference polynomial:

$$P'_2(3.5) = -0.084040 + [2(3.5) - (3.4 + 3.5)](0.023400) = -0.081700 \quad (5.14a)$$

$$P''_2(3.5) = 2(0.023400) = 0.046800 \quad (5.14b)$$

The results obtained by the three procedures are identical since the same three points are used in all three procedures. The error in $f'(3.5)$ is $\text{Error} = f'(3.5) - P'_2(3.5) = -0.081700 - (-0.081633) = -0.000067$, and the error in $f''(3.5)$ is $\text{Error} = f''(3.5) - P''_2(3.5) = 0.046800 - (0.046647) = 0.000153$.

5.3 EQUALLY SPACED DATA

When the tabular data to be differentiated are known at equally spaced points, the Newton forward-difference and backward-difference polynomials, presented in Section 4.6, can be fit to the discrete data with much less effort than a direct fit polynomial, a Lagrange polynomial, or a divided difference polynomial. This can significantly decrease the amount of effort required to evaluate derivatives. Thus,

$$f'(x) \cong \frac{d}{dx}(P_n(x)) = P'_n(x) \quad (5.15)$$

where $P_n(x)$ is either the Newton forward-difference or backward-difference polynomial.

5.3.1. Newton Forward-Difference Polynomial

Recall the Newton forward-difference polynomial, Eq. (4.88):

$$P_n(x) = f_0 + s \Delta f_0 + \frac{s(s-1)}{2} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{6} \Delta^3 f_0 + \dots + \text{Error} \quad (5.16)$$

$$\text{Error} = \binom{s}{n+1} h^{n+1} f^{(n+1)}(\xi) \quad x_0 \leq \xi \leq x_n \quad (5.17)$$

where the interpolating parameter s is given by

$$s = \frac{x - x_0}{h} \quad \rightarrow \quad x = x_0 + sh \quad (5.18)$$

Equation (5.15) requires that the approximating polynomial be an explicit function of x , whereas Eq. (5.16) is implicit in x . Either Eq. (5.16) must be made explicit in x by introducing Eq. (5.18) into Eq. (5.16), or the differentiation operations in Eq. (5.15) must be transformed into explicit operations in terms of s , so that Eq. (5.16) can be used directly.

The first approach leads to a complicated procedure, so the second approach is taken. From Eq. (5.18), $x = x(s)$. Thus,

$$f'(x) \cong \frac{d}{dx}(P_n(x)) = P'_n(x) = \frac{d}{ds}(P_n(s)) \frac{ds}{dx} \quad (5.19)$$

From Eq. (5.18),

$$\frac{ds}{dx} = \frac{1}{h} \quad (5.20)$$

Thus, Eq. (5.19) gives

$$f'(x) \cong P'_n(x) = \frac{1}{h} \frac{d}{ds}(P_n(s)) \quad (5.21)$$

Substituting Eq. (5.16) into Eq. (5.21) and differentiating gives

$$\begin{aligned} P'_n(x) &= \frac{1}{h} [\Delta f_0 + \frac{1}{2}[(s-1)+s]\Delta^2 f_0 + \frac{1}{6}[(s-1)(s-2)+s(s-2) \\ &\quad + s(s-1)]\Delta^3 f_0 + \dots] \end{aligned}$$

Simplifying Eq. (5.22) yields

$$P'_n(x) = \frac{1}{h} \left(\Delta f_0 + \frac{2s-1}{2} \Delta^2 f_0 + \frac{3s^2-6s+2}{6} \Delta^3 f_0 + \dots \right) \quad (5.23)$$

The second derivative is obtained as follows:

$$f''(x) \cong \frac{d}{dx}(P'_n(x)) = P''_n(x) = \frac{d}{ds}(P'_n(s)) \frac{ds}{dx} = \frac{1}{h} \frac{d}{ds}(P'_n(s)) \quad (5.24)$$

Substituting Eq. (5.23) into Eq. (5.24), differentiating, and simplifying yields

$$P''_n(x) = \frac{1}{h} \frac{d}{ds}(P'_n(s)) = \frac{1}{h^2} (\Delta^2 f_0 + (s-1) \Delta^3 f_0 + \dots) \quad (5.25)$$

Higher-order derivatives can be obtained in a similar manner. Recall that $\Delta^n f$ becomes less and less accurate as n increases. Consequently, higher-order derivatives become increasingly less accurate.

At $x = x_0$, $s = 0.0$, and Eqs. (5.23) and (5.25) becomes

$$P'_n(x_0) = \frac{1}{h} \left(\Delta f_0 - \frac{1}{2} \Delta^2 f_0 + \frac{1}{3} \Delta^3 f_0 - \frac{1}{4} \Delta^4 f_0 + \dots \right) \quad (5.26)$$

$$P''_n(x_0) = \frac{1}{h^2} (\Delta^2 f_0 - \Delta^3 f_0 + \dots) \quad (5.27)$$

Equations (5.26) and (5.27) are one-sided forward-difference formulas.

The error associated with numerical differentiation can be determined by differentiating the error term, Eq. (5.17). Thus,

$$\frac{d}{dx}(\text{Error}) = \frac{d}{ds} \left[\binom{s}{n+1} h^{n+1} f^{(n+1)}(\xi) \right] \frac{1}{h} \quad (5.28)$$

From the definition of the binomial coefficient, Eq. (4.90):

$$\binom{s}{n+1} = \frac{s(s-1)(s-2)\cdots(s-n)}{(n+1)!} \quad (5.29)$$

Substituting Eq. (5.29) into Eq. (5.28) and differentiating yields

$$\frac{d}{dx}(\text{Error}) = h^n f^{(n+1)}(\xi) \left[\frac{(s-1)(s-2)\cdots(s-n) + \cdots + s(s-1)\cdots(s-n+1)}{(n+1)!} \right] \quad (5.30)$$

At $x = x_0$, $s = 0.0$, and Eq. (5.30) gives

$$\frac{d}{dx}[\text{Error}(x_0)] = \frac{(-1)^n}{(n+1)} h^n f^{(n+1)}(\xi) \neq 0 \quad (5.31)$$

Even though there is no error in $P_n(x_0)$, there is error in $P'_n(x_0)$.

The order of an approximation is the rate at which the error of the approximation approaches zero as the interval h approaches zero. Equation (5.31) shows that the one-sided first derivative approximation $P'_n(x_0)$ is order n , which is written $O(h^n)$, when the polynomial $P_n(x)$ includes the n th forward difference. For example, $P'_1(x)$ is $O(h)$, $P'_2(x)$ is $O(h^2)$, etc. Each additional differentiation introduces another h into the denominator of the error term, so the order of the result drops by one for each additional differentiation. Thus, $P'_n(x_0)$ is $O(h^n)$, $P''_n(x_0)$ is $O(h^{n-1})$, etc.

A more direct way to determine the order of a derivative approximation is to recall that the error term of all difference polynomials is given by the first neglected term in the polynomial, with $\Delta^{(n+1)}f$ replaced by $h^{n+1}f^{(n+1)}(\xi)$. Each differentiation introduces an additional h into the denominator of the error term. For example, from Eqs. (5.26) and (5.27), if terms through Δ^2f_0 are accounted for, the error for $P'_2(x)$ is $O(h^3)/h = O(h^2)$ and the error for $P''_n(x)$ is $O(h^3)/h^2 = O(h)$. To achieve $O(h^2)$ for $P''_n(x_0)$, $P_3(x)$ must be used.

Example 5.2. Newton forward-difference polynomial, one-sided.

Let's solve the example problem presented in Section 5.1 using a Newton forward-difference polynomial with base point $x_0 = 3.5$, so $x_0 = 3.5$ in Eqs. (5.26) and (5.27). Selecting data points from Figure 5.1 and constructing the difference table gives

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$
3.5	0.285714			
3.6	0.277778	-0.007936		
3.7	0.270270	-0.007508	0.000428	-0.000032
3.8	0.263158	-0.007112	0.000396	

Substituting values into Eq. (5.26) gives

$$P'_n(3.5) = \frac{1}{0.1} \left[(-0.007936) - \frac{1}{2}(0.000428) + \frac{1}{3}(-0.000032) + \dots \right] \quad (5.32)$$

The order of the approximation of $P'_n(x)$ is the same as the order of the highest-order difference included in the evaluation. The first term in Eq. (5.32) is Δf_0 , so evaluating that term gives an $O(h)$ result. The second term in Eq. (5.32) is $\Delta^2 f_0$, so evaluating that term yields an $O(h^2)$ result, etc. Evaluating Eq. (5.32) term by term yields

$$\begin{aligned} P'_n(3.5) &= -0.07936 && \text{first order} && \text{Error} = 0.00227 \\ &= -0.08150 && \text{second order} && = 0.00013 \\ &= -0.08161 && \text{third order} && = 0.00002 \end{aligned}$$

The first-order result is quite inaccurate. The second- and third-order results are quite good. In all cases, only five significant digits after the decimal place are obtained.

Equation (5.27) gives

$$P''_n(3.5) = \frac{1}{(0.1)^2} [0.000428 - (-0.000032) + \dots] \quad (5.33)$$

The order of the approximation of $P''_n(x)$ is one less than the order of the highest-order difference included in the evaluation. The first term in Eq. (5.33) is $\Delta^2 f_0$, so evaluating that term gives an $O(h)$ result. The second term in Eq. (5.33) is $\Delta^3 f_0$, so evaluating that term yields an $O(h^2)$ result, etc. Evaluating Eq. (5.33) term by term yields

$$\begin{aligned} P''_n(3.5) &= 0.0428 && \text{first order} && \text{Error} = -0.0038 \\ &= 0.0460 && \text{second order} && = -0.0006 \end{aligned}$$

The first-order result is very poor. The second-order result, although much more accurate, has only four significant digits after the decimal place.

The results presented in this section illustrate the inherent inaccuracy associated with numerical differentiation. Equations (5.26) and (5.27) are both one-sided formulas. More accurate results can be obtained with centered differentiation formulas.

Centred-difference formulas can be obtained by evaluating the Newton forward-difference polynomial at points within the range of fit. For example, at $x = x_1$, $s = 1.0$, and Eqs. (5.23) and (5.25) give

$$P'_n(x_1) = \frac{1}{h} \left(\Delta f_0 + \frac{1}{2} \Delta^2 f_0 - \frac{1}{6} \Delta^3 f_0 + \dots \right) \quad (5.34)$$

$$P''_n(x_1) = \frac{1}{h^2} \left(\Delta^2 f_0 + \frac{1}{12} \Delta^4 f_0 + \dots \right) \quad (5.35)$$

From Eq. (5.34), $P'_1(x_1)$ is $O(h)$ and $P'_2(x_1)$ is $O(h^2)$, which is the same as the one-sided approximation, Eq. (5.26). However, $P''_2(x_1)$ is $O(h^2)$ since the $\Delta^3 f_0$ term is missing, whereas the one-sided approximation, Eq. (5.27), is $O(h)$. The increased order of the approximation of $P''_2(x_1)$ is due to centering the polynomial fit at point x_1 .

Example 5.3. Newton forward-difference polynomial, centered.

To illustrate a centered-difference formula, let's rework Example 5.2 using $x_0 = 3.4$ as the base point, so that $x_1 = 3.5$ is in the middle of the range of fit. Selecting data points from Figure 5.1 and constructing the difference table gives:

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$
3.4	0.294118	-0.008404		
3.5	0.285714	-0.007936	0.000468	-0.000040
3.6	0.277778	-0.007508	0.000428	
3.7	0.270270			

Substituting values into Eq. (5.34) gives

$$P'_n(3.5) = \frac{1}{0.1} \left[-0.008404 + \frac{1}{2}(0.000468) - \frac{1}{6}(-0.000040) + \dots \right] \quad (5.36)$$

Evaluating Eq. (5.36) term by term yields

$$\begin{aligned} P'_n(3.5) &= -0.08404 \text{--- first order} & \text{Error} &= -0.00241 \text{---} \\ &= -0.08170 \text{--- second order} & &= -0.00007 \text{---} \\ &= -0.08163 \text{--- third order} & &= 0.00000 \text{---} \end{aligned}$$

Equation (5.35) gives

$$P''_n(3.5) = \frac{1}{(0.1)^2} (0.000468 + \dots) \quad (5.37)$$

which yields

$$P''_n(3.5) = 0.0468 \text{--- second order} \quad \text{Error} = 0.0002 \text{---}$$

The error of the first-order result for $f'(3.5)$ is approximately the same magnitude as the error of the first-order result obtained in Example 5.2. The current result is a backward-difference approximation, whereas the result in Example 5.2 is a forward-difference approximation. The second-order centred-difference approximation of $f''(3.5)$ is more accurate than the second-order forward-difference approximation of $f''(3.5)$ in Example 5.2.

5.3.2. Newton Backward-Difference Polynomial

Recall the Newton backward-difference polynomial, Eq. (4.101):

$$P_n(x) = f_0 + s \nabla f_0 + \frac{s(s+1)}{2!} \nabla^2 f_0 + \frac{s(s+1)(s+2)}{3!} \nabla^3 f_0 + \dots \quad (5.38)$$

The first derivative $f'(x)$ is obtained from $P_n(x)$ as illustrated in Eq. (5.21):

$$f'(x) \cong P'_n(x) = \frac{1}{h} \frac{d}{ds} (P_n(s)) \quad (5.39)$$

Substituting Eq. (5.38) into Eq. (5.39), differentiating, and simplifying gives

$$P'_n(x) = \frac{1}{h} \left(\nabla f_0 + \frac{2s+1}{2} \nabla^2 f_0 + \frac{3s^2+6s+2}{6} \nabla^3 f_0 + \dots \right) \quad (5.40)$$

The second derivative $P''_n(x)$ is given by

$$P''_n(x) = \frac{1}{h^2} (\nabla^2 f_0 + (s+1) \nabla^3 f_0 + \dots) \quad (5.41)$$

Higher-order derivatives can be obtained in a similar manner. Recall that $\nabla^n f$ becomes less and less accurate as n increases. Consequently higher-order derivatives become increasingly less accurate.

At $x = x_0$, $s = 0.0$, and Eqs. (5.40) and (5.41) become

$$P'_n(x_0) = \frac{1}{h} \left(\nabla f_0 + \frac{1}{2} \nabla^2 f_0 + \frac{1}{3} \nabla^3 f_0 + \dots \right) \quad (5.42)$$

$$P''_n(x_0) = \frac{1}{h^2} (\nabla^2 f_0 + \nabla^3 f_0 + \dots) \quad (5.43)$$

Equations (5.42) and (5.43) are one-sided backward-difference formulas.

Centered-difference formulas are obtained by evaluating the Newton backward-difference polynomial at points within the range of fit. For example, at $x = x_{-1}$, $s = -1.0$, and Eqs. (5.40) and (5.41) give

$$P'_n(x_{-1}) = \frac{1}{h} \left(\nabla f_0 - \frac{1}{2} \nabla^2 f_0 - \frac{1}{6} \nabla^3 f_0 + \dots \right) \quad (5.44)$$

$$P''_n(x_{-1}) = \frac{1}{h^2} \left(\nabla^2 f_0 - \frac{1}{12} \nabla^4 f_0 + \dots \right) \quad (5.45)$$

The order of the derivative approximations are obtained by dividing the order of the first neglected term in each formula by the appropriate power of h , as discussed in Section 5.3.1 for derivative approximations based on Newton forward-difference polynomials.

5.3.3. Difference formulas

The formulas for derivatives developed in the previous subsections are expressed in terms of differences. Those formulas can be expressed directly in terms of function values if the order of the approximation is specified and the expressions for the differences in terms of function values are substituted into the formulas. The resulting formulas are called *difference formulas*. Several difference formulas are developed in this subsection to illustrate the procedure.

Consider the one-sided forward-difference formula for the first derivative, Eq. (5.26):

$$P'_n(x_0) = \frac{1}{h} \left(\Delta f_0 - \frac{1}{2} \Delta^2 f_0 + \frac{1}{3} \Delta^3 f_0 - \dots \right) \quad (5.46)$$

Recall that the error term associated with truncating the Newton forward-difference polynomial is obtained from the leading truncated term by replacing $\Delta^n f_0$ by $f^{(n)}(\xi)h^n$. Truncating Eq. (5.46) after Δf_0 gives

$$P'_n(x_0) = \frac{1}{h} [\Delta f_0 + 0(h^2)] \quad (5.47)$$

where $0(h^2)$ denotes the error term, and indicates the dependence of the error on the step size, h . Substituting $\Delta f_0 = (f_1 - f_0)$ into Eq. (5.47) yields

$$P'_1(x_0) = \frac{f_1 - f_0}{h} + 0(h) \quad (5.48)$$

Equation (5.48) is a one-sided first-order forward-difference formula for $f'(x_0)$. Truncating Eq. (5.46) after the $\Delta^2 f_0$ term gives

$$P'_2(x_0) = \frac{1}{h} \left(\Delta f_0 - \frac{1}{2} \Delta^2 f_0 + 0(h^3) \right) \quad (5.49)$$

Substituting Δf_0 and $\Delta^2 f_0$ into Eq. (5.49) and simplifying yields

$$P'_2(x_0) = \frac{-3f_0 + 4f_1 - f_2}{2h} + 0(h^2) \quad (5.50)$$

Higher-order difference formulas can be obtained in a similar manner.

Consider the one-sided forward-difference formula for the second derivative, Eq. (5.27). The following difference formulas can be obtained in the same manner that Eqs. (5.48) and (5.50) are developed.

$$P''_2(x_0) = \frac{f_0 - 2f_1 + f_2}{h^2} + 0(h) \quad (5.51)$$

$$P''_3(x_0) = \frac{2f_0 - 5f_1 + 4f_2 - f_3}{h^2} + 0(h^2) \quad (5.52)$$

Centered-difference formulas for $P'_n(x_1)$ and $P''_n(x_1)$ can be derived from Eqs. (5.34) and (5.35). Thus,

$$P'_2(x_1) = \frac{f_2 - f_0}{2h} + 0(h^2) \quad (5.53)$$

$$P''_2(x_1) = \frac{f_0 - 2f_1 + f_2}{h^2} + 0(h^2) \quad (5.54)$$

Difference formulas of any order can be developed in a similar manner for derivatives of any order, based on one-sided, centred, or nonsymmetrical differences. A selection of difference formulas is presented in Table 5.1.

Example 5.4. Newton polynomial difference formulas.

Let's illustrate the use of difference formulas obtained from Newton polynomials by solving the example problem presented in Section 5.1. Calculate the second-order centered-difference approximation of $f'(3.5)$ and $f''(3.5)$ using Eqs. (5.53) and (5.54). Thus,

$$P'_2(3.5) = \frac{f(3.6) - f(3.4)}{2(0.1)} = \frac{0.277778 - 0.294118}{2(0.1)} = -0.08170 \quad (5.55)$$

$$\begin{aligned} P''_2(3.5) &= \frac{f(3.6) - 2f(3.5) + f(3.4)}{(0.1)^2} = \frac{0.277778 - 2(0.285714) + 0.294118}{(0.1)^2} \\ &= 0.0468 \end{aligned} \quad (5.56)$$

These results are identical to the second-order results obtained in Example 5.3.

5.4 TAYLOR SERIES APPROACH

Difference formulas can also be developed using Taylor series. This approach is especially useful for deriving finite difference approximations of exact derivatives (both total derivatives and partial derivatives) that appear in differential equations.

Difference formulas for functions of a single variable, for example, $f(x)$, can be developed from the Taylor series for a function of a single variable, Eq. (0.6):

$$f(x) = f_0 + f'_0 \Delta x + \frac{1}{2} f''_0 \Delta x^2 + \cdots + \frac{1}{n!} f_0^{(n)} \Delta x^n + \cdots \quad (5.57)$$

where $f_0 = f(x_0)$, $f'_0 = f'(x_0)$, etc. The continuous spatial domain $D(x)$ must be discretized into an equally spaced grid of discrete points, as illustrated in Figure 5.4. For the discretized x space,

$$f(x_i) = f_i \quad (5.58)$$

where the subscript i denotes a particular spatial location. The Taylor series for $f(x)$ at grid points surrounding point i can be combined to obtain difference formulas for $f'(x_i)$, $f''(x_i)$, etc.

Difference formulas for functions of time, $f(t)$, can be developed from the Taylor series for $f(t)$:

$$f(t) = f_0 + f'_0 \Delta t + \frac{1}{2} f''_0 \Delta t^2 + \cdots + \frac{1}{n!} f_0^{(n)} \Delta t^n + \cdots \quad (5.59)$$

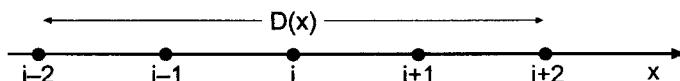


Figure 5.4 Continuous spatial domain $D(x)$ and discretized x space.

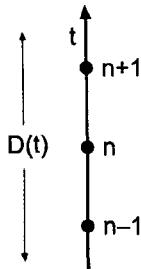


Figure 5.5 Continuous temporal domain $D(t)$ and discretized t space.

where $f_0 = f(t^0), f'_0 = f'(t^0)$, etc. The continuous temporal domain $D(t)$ must be discretized into a grid of discrete points, as illustrated in Figure 5.5. For the discretized t space,

$$f(t^n) = f^n \quad (5.60)$$

where the superscript n denotes a particular temporal location. The Taylor series for $f(t)$ at grid points surrounding grid point n can be combined to obtain difference formulas for $f'(t^n), f''(t^n)$, etc.

Difference formulas for functions of several variables, for example, $f(x, t)$, can be developed from the Taylor series for a function of several variables, Eq. (0.12):

$$\begin{aligned} f(x, t) = & f_0 + (f_x|_0 \Delta x + f_t|_0 \Delta t) + \frac{1}{2!} (f_{xx}|_0 \Delta x^2 + 2f_{xt}|_0 \Delta x \Delta t + f_{tt}|_0 \Delta t^2) \\ & + \cdots + \frac{1}{n!} \left(\Delta x \frac{\partial}{\partial x} + \Delta t \frac{\partial}{\partial t} \right)^n f_0 + \cdots \end{aligned} \quad (5.61)$$

where $f_0 = f(x_0, t_0), f_x|_0 = f_x(x_0, t_0)$, etc. The expression $(\cdots)^n$ is expanded by the binomial expansion, the increments in Δx and Δt are raised to the indicated powers, and terms such as $(\partial/\partial x)^n$, etc., are interpreted as $\partial^n/\partial x^n$, etc. The continuous xt domain, $D(x, t)$, must be discretized into an orthogonal equally spaced grid of discrete points, as illustrated in Figure 5.6. For the discrete grid,

$$f(x_i, t^n) = f_i^n \quad (5.62)$$

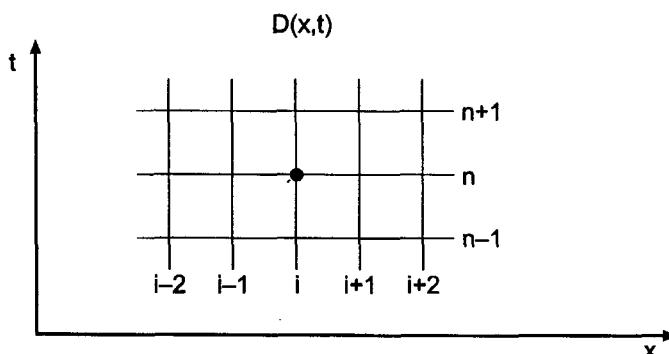


Figure 5.6 Continuous xt domain $D(x, t)$ and discretized xt space.

where the subscript i denotes a particular spatial location and the superscript n denotes a particular time level. The Taylor series for $f(x, t)$ at grid points surrounding point (i, n) can be combined to obtain difference formulas for f_x, f_t, f_{xt} , etc.

For partial derivatives of $f(x, t)$ with respect to x , $t = t_0 = \text{constant}$, $\Delta t = 0$, and Eq. (5.61) becomes

$$f(x, t_0) = f_0 + f_x|_0 \Delta x + \frac{1}{2} f_{xx}|_0 \Delta x^2 + \cdots + \frac{1}{n!} f_{(n)x}|_0 \Delta x^n + \cdots \quad (5.63)$$

Equation (5.63) is identical in form to Eq. (5.57), where f'_0 corresponds to $f_x|_0$, etc. The partial derivative $f_x|_0$ of the function $f(x, t)$ can be obtained from Eq. (5.63) in exactly the same manner as the total derivative, f'_0 , of the function $f(x)$ is obtained from Eq. (5.57). Since Eqs. (5.57) and (5.63) are identical in form, the difference formulas for f'_0 and $f_x|_0$ are identical if the same discrete grid points are used to develop the difference formulas. Consequently, difference formulas for partial derivatives of a function of several variables can be derived from the Taylor series for a function of a single variable. To emphasize this concept, the following common notation for derivatives will be used in the development of difference formulas for total derivatives and partial derivatives:

$$\frac{d}{dx}(f(x)) = f_x \quad (5.64)$$

$$\frac{\partial}{\partial x}(f(x, t)) = f_x \quad (5.65)$$

In a similar manner, partial derivatives of $f(x, t)$ with respect to t with $x = x_0 = \text{constant}$ can be obtained from the expression

$$f(x_0, t) = f_0 + f_t|_0 \Delta t + \frac{1}{2} f_{tt}|_0 \Delta t^2 + \cdots + \frac{1}{n!} f_{(n)t}|_0 \Delta t^n + \cdots \quad (5.66)$$

Partial derivatives of $f(x, t)$ with respect to t are identical in form to total derivatives of $f(t)$ with respect to t .

This approach does not work for mixed partial derivatives, such as f_{xt} . Difference formulas for mixed partial derivatives must be determined directly from the Taylor series for several variables, Eq. (5.61).

The Taylor series for the function $f(x)$, Eq. (5.57), can be written as

$$f(x) = f_0 + f_x|_0 \Delta x + \frac{1}{2} f_{xx}|_0 \Delta x^2 + \cdots + \frac{1}{n!} f_{(n)x}|_0 \Delta x^n + \cdots \quad (5.67)$$

The Taylor formula with remainder is given by Eq. (0.10):

$$f(x) = f_0 + f_x|_0 \Delta x + \frac{1}{2} f_{xx}|_0 \Delta x^2 + \cdots + \frac{1}{n!} f_{(n)x}|_0 \Delta x^n + R^{n+1} \quad (5.68)$$

where the remainder term R^{n+1} is given by

$$R^{n+1} = \frac{1}{(n+1)!} f_{(n+1)x}(\xi) \Delta x^{n+1} \quad (5.69)$$

where $x_0 \leq \xi \leq x_0 + \Delta x$.

The infinite Taylor series, Eq. (5.67), and the Taylor formula with remainder, Eq. (5.68), are equivalent. The error incurred by truncating the infinite Taylor series after the n th derivative is exactly the remainder term of the n th-order Taylor formula. Truncating the Taylor series is equivalent to dropping the remainder term of the Taylor formula. Finite difference approximations of exact derivatives can be obtained by solving for the exact derivative from either the infinite Taylor series or the Taylor formula, and then either

truncating the Taylor series or dropping the remainder term of the Taylor formula. These two procedures are identical. The terms which are truncated from the infinite Taylor series, which are identical to the remainder term of the Taylor formula, are called the *truncation error* of the finite difference approximation of the exact derivative. In most cases, our main concern is the *order* of the truncation error, which is the rate at which the truncation error approaches zero as $\Delta x \rightarrow 0$. The order of the truncation error, which is the order of the remainder term, is denoted by the notation $O(\Delta x^n)$.

Consider the equally spaced discrete finite difference grid illustrated in Figure 5.4. Choose point i as the base point, and write the Taylor series for f_{i+1} and f_{i-1} :

$$f_{i+1} = f_i + f_x|_i \Delta x + \frac{1}{2} f_{xx}|_i \Delta x^2 + \frac{1}{6} f_{xxx}|_i \Delta x^3 + \frac{1}{24} f_{xxxx}|_i \Delta x^4 + \dots \quad (5.70)$$

$$f_{i-1} = f_i - f_x|_i \Delta x + \frac{1}{2} f_{xx}|_i \Delta x^2 - \frac{1}{6} f_{xxx}|_i \Delta x^3 + \frac{1}{24} f_{xxxx}|_i \Delta x^4 - \dots \quad (5.71)$$

Subtracting Eq. (5.71) for f_{i-1} from Eq. (5.70) for f_{i+1} gives

$$f_{i+1} - f_{i-1} = 2f_x|_i \Delta x + \frac{1}{3} f_{xxx}|_i \Delta x^3 + \dots \quad (5.72)$$

Letting the f_{xxx} term be the remainder term and solving for $f_x|_i$ yields

$$f_x|_i = \frac{f_{i+1} - f_{i-1}}{2 \Delta x} - \frac{1}{6} f_{xxx}(\xi) \Delta x^2 \quad (5.73)$$

where $x_{i-1} \leq \xi \leq x_{i+1}$. Equation (5.73) is an exact expression for $f_x|_i$. If the remainder term is truncated, which is equivalent to truncating the infinite Taylor series, Eqs. (5.70) and (5.71), Eq. (5.73) yields an $O(\Delta x^2)$ finite difference approximation of $f_x|_i$. Thus,

$$f_x|_i = \frac{f_{i+1} - f_{i-1}}{2 \Delta x} \quad (5.74)$$

The truncated result is identical to the result obtained from the Newton forward-difference polynomial, Eq. (5.53).

Adding Eq. (5.70) for f_{i+1} and Eq. (5.71) for f_{i-1} gives

$$f_{i+1} + f_{i-1} = 2f_i + f_{xx}|_i \Delta x^2 + \frac{1}{12} f_{xxxx}|_i \Delta x^4 + \dots \quad (5.75)$$

Letting the f_{xxxx} term be the remainder term and solving for $f_{xx}|_i$ yields

$$f_{xx}|_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{2 \Delta x} - \frac{1}{12} f_{xxxx}(\xi) \Delta x^2 \quad (5.76)$$

where $x_{i-1} \leq \xi \leq x_{i+1}$. Truncating the remainder term yields a finite difference approximation for $f_{xx}|_i$. Thus,

$$f_{xx}|_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} \quad (5.77)$$

The truncated result is identical to the result obtained from the Newton forward-difference polynomial, Eq. (5.54).

Equations (5.74) and (5.77) are centered-difference formulas. They are inherently more accurate than one-sided difference formulas.

Example 5.5. Taylor series difference formulas.

Let's illustrate the use of difference formulas obtained from Taylor series by evaluating $f'(3.5)$ and $f''(3.5)$ for the data presented in Figure 5.1 using Eqs. (5.74) and (5.77), respectively. To obtain the most accurate results possible, use the closest data points to $x = 3.5$, that is, $x = 3.4$ and 3.6 . Thus, from Eq. (5.74),

$$f'(3.5) = \frac{f(3.6) - f(3.4)}{2(0.1)} = \frac{0.279778 - 0.294118}{2(0.1)} = -0.08170 \quad (5.78)$$

From Eq. (5.77),

$$\begin{aligned} f''(3.5) &= \frac{f(3.6) - 2.0f(3.5) + f(3.4)}{2.0(0.1)} = \frac{0.279778 - 2.0(0.285714) + 0.294118}{2.0(0.1)} \\ &= 0.0468 \end{aligned} \quad (5.79)$$

These are the same results that are obtained with the difference formulas developed from the Newton forward-difference polynomials illustrated in Example 5.4, Eqs. (5.55) and (5.56).

Equations (5.74) and (5.77) are difference formulas for spatial derivatives. Difference formulas for time derivatives can be developed in a similar manner. The time dimension can be discretized into a discrete temporal grid, as illustrated in Figure 5.5, where the superscript n denotes a specific value of time. Thus, $f(t^n) = f^n$. Choose point n as the base point, and write the Taylor series for f^{n+1} and f^{n-1} :

$$f^{n+1} = f^n + f_t|_n^n \Delta t + \frac{1}{2} f_{tt}|_n^n \Delta t^2 + \dots \quad (5.80)$$

$$f^{n-1} = f^n - f_t|_n^n \Delta t + \frac{1}{2} f_{tt}|_n^n \Delta t^2 - \dots \quad (5.81)$$

Letting the $f_{tt}|_n^n$ term be the remainder term and solving Eq. (5.80) for $f_t|_n^n$ yields

$$f_t|_n^n = \frac{f^{n+1} - f^n}{\Delta t} - \frac{1}{2} f_{tt}(\tau) \Delta t \quad (5.82)$$

where $t^n \leq \tau \leq t^{n+1}$. Equation (5.82) is a first-order forward-difference formula for $f_t|_n^n$. Subtracting Eq. (5.81) for f^{n-1} from Eq. (5.80) for f^{n+1} gives

$$f^{n+1} - f^{n-1} = 2f_t|_n^n \Delta t + \frac{1}{3} f_{ttt}|_n^n \Delta t^3 + \dots \quad (5.83)$$

Letting the $f_{ttt}|_n^n$ term be the remainder term and solving for $f_t|_n^n$ yields

$$f_t|_n^n = \frac{f^{n+1} - f^{n-1}}{2 \Delta t} - \frac{1}{6} f_{ttt}(\bar{\tau}) \Delta t^2 \quad (5.84)$$

where $t^{n-1} \leq \bar{\tau} \leq t^{n+1}$. Equation (5.84) is a second-order centred-difference formula for $f_t|_t^n$. Centred-difference formulas are inherently more accurate than one-sided difference formulas, such as Eq. (5.82).

Difference formulas of any order, based on one-sided forward differences, one-sided backward differences, centered differences, nonsymmetrical differences, etc., can be obtained by different combinations of the Taylor series for $f(x)$ or $f(t)$ at various grid points. Higher-order difference formulas require more grid points, as do formulas for higher-order derivatives.

Example 5.6. Third-order nonsymmetrical difference formula for f_x .

Let's develop a third-order, nonsymmetrical, backward-biased, difference formula for f_x . The Taylor series for $f(x)$ is:

$$f(x) = f_i + f_x|_i \Delta x + \frac{1}{2} f_{xx}|_i \Delta x^2 + \frac{1}{6} f_{xxx}|_i \Delta x^3 + \frac{1}{24} f_{xxxx}|_i \Delta x^4 + \dots \quad (5.85)$$

Three questions must be answered before the difference formula can be developed: (a) What is to be the order of the remainder term, (b) how many grid points are required, and (c) which grid points are to be used? The coefficient of f_x is Δx . If the remainder term in the difference formula is to be $O(\Delta x^3)$, then the remainder term in the Taylor series must be $O(\Delta x^4)$, since the formulas must be divided by Δx when solving for f_x . Consequently, three grid points, in addition to the base point i , are required, so that f_{xx} and f_{xxx} can be eliminated from the Taylor series expansions, thus giving a third-order difference formula for f_x . For a backward-biased difference formula, choose grid points $i+1$, $i-1$, and $i-2$.

The fourth-order Taylor series for f_{i+1} , f_{i-1} , and f_{i-2} are:

$$f_{i+1} = f_i + f_x|_i \Delta x + \frac{1}{2} f_{xx}|_i \Delta x^2 + \frac{1}{6} f_{xxx}|_i \Delta x^3 + \frac{1}{24} f_{xxxx}(\xi_1) \Delta x^4 + \dots \quad (5.86)$$

$$f_{i-1} = f_i - f_x|_i \Delta x + \frac{1}{2} f_{xx}|_i \Delta x^2 - \frac{1}{6} f_{xxx}|_i \Delta x^3 + \frac{1}{24} f_{xxxx}(\xi_{-1}) \Delta x^4 - \dots \quad (5.87)$$

$$f_{i-2} = f_i - 2f_x|_i \Delta x + \frac{4}{2} f_{xx}|_i \Delta x^2 - \frac{8}{6} f_{xxx}|_i \Delta x^3 + \frac{16}{24} f_{xxxx}(\xi_{-2}) \Delta x^4 - \dots \quad (5.88)$$

Forming the combination $(f_{i+1} - f_{i-1})$ gives

$$(f_{i+1} - f_{i-1}) = 2f_x|_i \Delta x + \frac{2}{6} f_{xxx}|_i \Delta x^3 + O(\Delta x^5) \quad (5.89)$$

Forming the combination $(4f_{i+1} - f_{i-2})$ gives

$$(4f_{i+1} - f_{i-2}) = 3f_i + 6f_x|_i \Delta x + \frac{12}{6} f_{xxx}|_i \Delta x^3 - \frac{12}{24} f_{xxxx}(\bar{\xi}) \Delta x^4 + O(\Delta x^5) \quad (5.90)$$

where $x_{i-2} \leq \bar{\xi} \leq x_{i+1}$. Multiplying Eq. (5.89) by 6 and subtracting Eq. (5.90) gives

$$6(f_{i+1} - f_{i-1}) - (4f_{i+1} - f_{i-2}) = -3f_i + 6f_x|_i \Delta x + \frac{12}{24} f_{xxxx}(\bar{\xi}) \Delta x^4 \quad (5.91)$$

Solving Eq. (5.91) for $f_x|_i$ yields

$$f_x|_i = \frac{f_{i-2} - 6f_{i-1} + 3f_i + 2f_{i+1}}{6\Delta x} - \frac{1}{2} f_{xxxx}(\bar{\xi}) \Delta x^3$$

(5.92)

Truncating Eq. (5.92) yields a third-order, nonsymmetrical, backward-biased, difference formula for $f_x|_i$.

In summary, the procedure for developing difference formulas by the Taylor series approach is as follows.

1. Specify the order n of the derivative $f_{(n)x}$ for which the difference formula is to be developed.
2. Choose the order m of the remainder term in the difference formula Δx^m .
3. Determine the order of the remainder term in the Taylor series, Δx^{m+n} .
4. Specify the type of difference formula desired: centered, forward, backward, or nonsymmetrical.
5. Determine the number of grid points required, which is at most $(m + n - 1)$.
6. Write the Taylor series of order $(m + n)$ at the $(m + n - 1)$ grid points.
7. Combine the Taylor series to eliminate the undesired derivatives, and solve for the desired derivative and the leading truncation error term.

For temporal derivatives, replace x by t in steps 1 to 7.

5.5 DIFFERENCE FORMULAS

Table 5.1 presents several difference formulas for both time derivatives and space derivatives. These difference formulas are used extensively in Chapters 7 and 8 in the numerical solution of ordinary differential equations, and in Chapters 9 to 11 in the numerical solution of partial differential equations.

5.6 ERROR ESTIMATION AND EXTRAPOLATION

When the functional form of the error of a numerical algorithm is known, the error can be estimated by evaluating the algorithm for two different increment sizes. The error estimate can be used both for error control and extrapolation.

Consider a numerical algorithm which approximates an exact calculation with an error that depends on an increment, h . Thus,

$$f_{\text{exact}} = f(h) + Ah^n + Bh^{n+m} + Ch^{n+2m} + \dots \quad (5.113)$$

where n is the order of the leading error term and m is the increment in the order of the following error terms. Applying the algorithm at two increment sizes, $h_1 = h$ and $h_2 = h/R$, gives

$$f_{\text{exact}} = f(h) + Ah^n + O(h^{n+m}) \quad (5.114a)$$

$$f_{\text{exact}} = f(h/R) + A(h/R)^n + O(h^{n+m}) \quad (5.114b)$$

Subtracting Eq. (5.114b) from Eq. (5.114a) gives

$$0 = f(h) - f(h/R) + Ah^n + A(h/R)^n + O(h^{n+m}) \quad (5.115)$$

Table 5.1. Difference Formulas

$$f_t|_t^n = \frac{f^{n+1} - f^n}{\Delta t} - \frac{1}{2} f_{tt}(\tau) \Delta t \quad (5.93)$$

$$f_t|_t^n = \frac{f^n - f^{n-1}}{\Delta t} + \frac{1}{2} f_{tt}(\tau) \Delta t \quad (5.94)$$

$$f_t|_t^{n+1/2} = \frac{f^{n+1} - f^n}{\Delta t} - \frac{1}{24} f_{ttt}(\tau) \Delta t^2 \quad (5.95)$$

$$f_t|_t^n = \frac{f^{n+1} - f^{n-1}}{2 \Delta t} - \frac{1}{6} f_{ttt}(\tau) \Delta t^2 \quad (5.96)$$

$$f_x|_i = \frac{f_{i+1} - f_i}{\Delta x} - \frac{1}{2} f_{xx}(\xi) \Delta x \quad (5.97)$$

$$f_x|_i = \frac{f_i - f_{i-1}}{\Delta x} + \frac{1}{2} f_{xx}(\xi) \Delta x \quad (5.98)$$

$$f_x|_i = \frac{f_{i+1} - f_{i-1}}{2 \Delta x} - \frac{1}{6} f_{xxx}(\xi) \Delta x^2 \quad (5.99)$$

$$f_x|_i = \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2 \Delta x} - \frac{1}{3} f_{xxx}(\xi) \Delta x^2 \quad (5.100)$$

$$f_x|_i = \frac{f_{i-2} - 4f_{i-1} + 3f_i}{2 \Delta x} + \frac{1}{3} f_{xxx}(\xi) \Delta x^2 \quad (5.101)$$

$$f_x|_i = \frac{-11f_i + 18f_{i+1} - 9f_{i+2} + 2f_{i+3}}{6 \Delta x} - \frac{1}{4} f_{xxxx}(\xi) \Delta x^3 \quad (5.102)$$

$$f_x|_i = \frac{-2f_{i-3} + 9f_{i-2} - 18f_{i-1} + 11f_i}{6 \Delta x} + \frac{1}{4} f_{xxxx}(\xi) \Delta x^3 \quad (5.103)$$

$$f_x|_i = \frac{f_{i-2} - 6f_{i-1} + 3f_i + 2f_{i+1}}{6 \Delta x} - \frac{1}{12} f_{xxxx}(\xi) \Delta x^3 \quad (5.104)$$

$$f_x|_i = \frac{-2f_{i-1} - 3f_i + 6f_{i+1} - f_{i+2}}{6 \Delta x} + \frac{1}{12} f_{xxxx}(\xi) \Delta x^3 \quad (5.105)$$

$$f_x|_i = \frac{f_{i-2} - 8f_{i-1} + 8f_{i+1} - f_{i+2}}{12 \Delta x} + \frac{1}{30} f_{xxxxx}(\xi) \Delta x^4 \quad (5.106)$$

$$f_{xx}|_i = \frac{f_i - 2f_{i+1} + f_{i+2}}{\Delta x^2} - f_{xxx}(\xi) \Delta x \quad (5.107)$$

$$f_{xx}|_i = \frac{f_{i-2} - 2f_{i-1} + f_i}{\Delta x^2} + f_{xxx}(\xi) \Delta x \quad (5.108)$$

$$f_{xx}|_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2} - \frac{1}{12} f_{xxxx}(\xi) \Delta x^2 \quad (5.109)$$

$$f_{xx}|_i = \frac{2f_i - 5f_{i+1} + 4f_{i+2} - f_{i+3}}{\Delta x^2} + \frac{11}{12} f_{xxxx}(\xi) \Delta x^2 \quad (5.110)$$

$$f_{xx}|_i = \frac{-f_{i-3} + 4f_{i-2} - 5f_{i-1} + 2f_i}{\Delta x^2} + \frac{11}{12} f_{xxxx}(\xi) \Delta x^2 \quad (5.111)$$

$$f_{xx}|_i = \frac{-f_{i-2} + 16f_{i-1} - 30f_i + 16f_{i+1} - f_{i+2}}{12 \Delta x^2} + \frac{1}{90} f_{xxxxx}(\xi) \Delta x^4 \quad (5.112)$$

Solving Eq. (5.115) for the leading error terms in Eqs. (5.114a) and (5.114b) yields

$$\text{Error}(h) = Ah^n = \frac{R^n}{R^n - 1} (f(h/R) - f(h)) \quad (5.116a)$$

$$\text{Error}(h/R) = A(h/R)^n = \frac{1}{R^n - 1} (f(h/R) - f(h)) \quad (5.116b)$$

Equation (5.116) can be used to estimate the leading error terms in Eq. (5.114).

The error estimates can be added to the approximate results to yield an improved approximation. This process is called *extrapolation*. Adding Eq. (5.116b) to Eq. (5.114b) gives

$$\text{Extrapolated value} = f(h/R) + \frac{1}{R^n - 1} (f(h/R) - f(h)) + 0(h^{n+m}) \quad (5.117)$$

The error of the extrapolated value is $0(h^{n+m})$. Two $0(h^{n+m})$ extrapolated results can be extrapolated to give an $0(h^{n+2m})$ result, where the exponent, n , in Eq. (5.117) is replaced with the exponent, $n + m$. Higher-order extrapolations can be obtained by successive applications of Eq. (5.117).

Example 5.7. Error estimation and extrapolation.

Example 5.5 evaluates $f'(3.5)$ using Eq. (5.73) with $\Delta x = 0.1$. A more accurate result could be obtained by evaluating Eq. (5.73) with $\Delta x = 0.05$, which requires data at $x = 3.45$ and 3.55 . As seen in Figure 5.1, those points are not available. However, data are available at $x = 3.3$ and 3.7 , for which $\Delta x = 0.2$. Applying Eq. (5.73) with $\Delta x = 0.2$ gives

$$f'(3.5) = \frac{f(3.7) - f(3.3)}{2(0.2)} = \frac{0.270270 - 0.303030}{2(0.2)} = -0.081900 \quad (5.118)$$

The exact error in this result is $\text{Error} = -0.081900 - (-0.081633) = -0.000267$, which is approximately four times larger than the exact error obtained in Example 5.5 where $f'(3.5) = -0.081700$, for which the exact error is $\text{Error} = -0.081700 - (-0.081633) = -0.000067$.

Now that two estimates of $f'(3.5)$ are available, the error estimate for the result with the smaller Δx can be calculated from Eq. (5.116b). Thus,

$$\text{Error}(\Delta x/2) = \frac{1}{3} [-0.081700 - (-0.081900)] = 0.000067 \quad (5.119)$$

Applying the extrapolation formula, Eq. (5.117), gives

$$\text{Extrapolated value} = -0.081700 + 0.000067 = -0.081633 \quad (5.120)$$

which is the exact value to six digits after the decimal place. The value of error estimation and extrapolation are obvious in this example.

5.7 PROGRAMS

Two procedures for numerical differentiation of tabular data are presented in this section:

1. Derivatives of unequally spaced data
2. Derivatives of equally spaced data

All of the subroutines are written for a *quadratic* polynomial. Higher-degree polynomials can be constructed by following the patterns of the quadratic polynomials. The variable *ndeg*, which specifies the degree of the polynomials, is thus not needed in the present programs. It is included, however, to facilitate the addition of other-degree polynomials to the subroutines, in which case the degree of the polynomial to be evaluated must be specified.

The basic computational algorithms are presented as completely self-contained subroutines suitable for use in other programs. Input data and output statements are contained in a main (or driver) program written specifically to illustrate the use of each subroutine.

5.7.1. Derivatives of Unequally Spaced Data

Three procedures are presented for differentiating unequally spaced data:

1. Differentiation of a direct fit polynomial
2. Differentiation of a Lagrange polynomial
3. Differentiation of a divided difference polynomial

FORTRAN subroutines *direct*, *lagrange*, and *divdiff* are presented in this subsection for implementing these procedures. A common *program main* defines the data sets and prints them, calls one of the subroutines to implement the solution, and prints the solution. The only change in *program main* for the three subroutines is the *call* statement to the particular subroutine and the output *format* statement identifying the particular subroutine.

5.7.1.1 Direct Fit Polynomial

The first- and second-order derivatives of a direct fit polynomial are given by Eq. (5.7):

$$P'_n(x) = a_1 + 2a_2x + 3a_3x^2 + \cdots + na_nx^{n-1} \quad (5.121a)$$

$$P''_n(x) = 2a_2 + 6a_3x + \cdots + n(n-1)a_nx^{n-2} \quad (5.121b)$$

A FORTRAN subroutine, *subroutine direct*, for implementing Eq. (5.121) for a quadratic direct fit polynomial is presented in Program 5.1.

Program 5.1. Differentiation of a direct fit polynomial program.

```

program main
c   main program to illustrate numerical diff. subroutines
c   ndim  array dimension, n = 3 in this example
c   ndeg  degree of polynomial, ndeg = 2 in this example
c   n     number of data points
c   xp    value of x at which to evaluate the derivatives
c   x     independent variable, x(i)

```

```

c      f      dependent variable, f(i)
c      fx      numerical approximation of first derivative
c      fxx     numerical approximation of second derivative
dimension x(3),f(3),a(3,3),b(3),c(3)
data ndim,ndeg,n,xp / 3, 2, 3, 3.5 /
data (x(i),i=1,3) / 3.4, 3.5, 3.6 /
data (f(i),i=1,3) / 0.294118, 0.285714, 0.277778 /
write (6,1000)
do i=1,n
    write (6,1010) i,x(i),f(i)
end do
call direct (ndim,ndeg,n,x,f,xp,fx,fxx,a,b,c)
write (6,1020) fx,fxx
stop
1000 format (' Direct fit polynomial'// ' ' i',6x,'x',11x,'f'// ' ')
1010 format (i3,5f12.6)
1020 format (' // fx =',f12.6,' and fxx =',f12.6)
end

subroutine direct (ndim,ndeg,n,x,f,xp,fx,fxx,a,b,c)
c      direct fit polynomial differentiation
dimension x(ndim),f(ndim),a(ndim,ndim),b(ndim),c(ndim)
do i=1,n
    a(i,1)=1.0
    a(i,2)=x(i)
    a(i,3)=x(i)**2
    b(i)=f(i)
end do
call gauss (ndim,n,a,b,c)
fx=c(2)+2.0*c(3)*xp
fxx=2.0*c(3)
return
end

subroutine gauss (ndim,n,a,b,x)
c      implements simple gauss elimination
end

```

The data set used to illustrate *subroutine direct* is taken from Example 5.1. The output generated by the program is presented in Output 5.1.

Output 5.1. Solution by differentiation of a direct fit polynomial.

Direct fit polynomial

i	x	f
1	3.400000	0.294118
2	3.500000	0.285714
3	3.600000	0.277778

fx = -0.081700 and fxx = 0.046800

5.7.1.2 Lagrange Polynomial

The first- and second-order derivatives of a quadratic Lagrange polynomial are given by Eq. (5.9):

$$f'(x) \cong P'_2(x) = \frac{2x - (b + c)}{(a - b)(a - c)} f(a) + \frac{2x - (a + c)}{(b - a)(b - c)} f(b) + \frac{2x - (a + b)}{(c - a)(c - b)} f(c) \quad (5.122a)$$

$$f''(x) \cong P''_2(x) = \frac{2f(a)}{(a - b)(a - c)} + \frac{2f(b)}{(b - a)(b - c)} + \frac{2f(c)}{(c - a)(c - b)} \quad (5.122b)$$

A FORTRAN subroutine, *subroutine lagrange*, for implementing Eq. (5.122) is presented in Program 5.2.

Program 5.2. Differentiation of a Lagrange polynomial program.

```

program main
c      main program to illustrate numerical diff. subroutines
      dimension x(3),f(3)
      call lagrange (ndim,ndeg,n,x,f,xp,fx,fxx,a,b,c)
1000 format (' Lagrange polynomial'// ' ' i',6x,'x',11x,'f'/' ')
      end

      subroutine lagrange (ndim,ndeg,n,x,f,xp,fx,fxx)
c      Lagrange polynomial differentiation
      dimension x(ndim),f(ndim)
      a=x(1)
      b=x(2)
      c=x(3)
      fx=(2.0*xp-(b+c))*f(1)/(a-b)/(a-c)+(2.0*xp-(a+c))*f(2)/(b-a)
      1 /(b-c)+(2.0*xp-(a+b))*f(3)/(c-a)/(c-b)
      fxx=2.0*f(1)/(a-b)/(a-c)+2.0*f(2)/(b-a)/(b-c)+2.0*f(3)/(c-a)
      1 /(c-b)
      return
      end

```

The data set used to illustrate *subroutine Lagrange* is taken from Example 5.1. The output generated by the program is presented in Output 5.2.

Output 5.2. Solution by differentiation of a Lagrange polynomial.

Lagrange polynomial

i	x	f
1	3.400000	0.294118
2	3.500000	0.285714
3	3.600000	0.277778

fx = -0.081700 and *fxx* = 0.046800

5.7.1.3. Divided Difference Polynomial

The first- and second-order derivatives of a divided difference polynomial are given by Eq. (5.11):

$$\begin{aligned} f'(x) \cong P'_n(x) = & f_i^{(1)} + [2x - (x_0 + x_1)] f_i^{(2)} \\ & + [3x^2 - 2(x_0 + x_1 + x_2)x + (x_0x_1 + x_0x_2 + x_1x_2)] f_i^{(3)} + \dots \end{aligned} \quad (5.123a)$$

$$f''(x) \cong P''_n(x) = 2f_i^{(2)} + [6x - 2(x_0 + x_1 + x_2)] f_i^{(3)} + \dots \quad (5.123b)$$

A FORTRAN subroutine, *subroutine divdiff*, for implementing Eq. (5.123) for $P_2(x)$ is presented in Program 5.3.

Program 5.3. Differentiation of a divided difference polynomial program.

```

program main
c      main program to illustrate numerical diff. subroutines
dimension x(3),f(3)
call divdiff (ndim,ndeg,n,x,f,xp,fx,fxx,a,b,c)
1000 format (' Divided diff. poly.'// ' //   i',6x,'x',11x,'f'//  ')
end

subroutine divdiff (ndim,ndeg,n,x,f,xp,fx,fxx)
c      divided difference polynomial differentiation
dimension x(ndim),f(ndim)
f11=(f(2)-f(1))/(x(2)-x(1))
f21=(f(3)-f(2))/(x(3)-x(2))
f12=(f21-f11)/(x(3)-x(1))
fx=f11+(2.0*xp-x(1)-x(2))*f12
fxx=2.0*f12
return
end

```

The data set used to illustrate *subroutine divdiff* is taken from Example 5.1. The output generated by the program is presented in Output 5.3.

Output 5.3. Solution by differentiation of a divided difference polynomial.

Divided diff. polynomial

i	x	f
1	3.400000	0.294118
2	3.500000	0.285714
3	3.600000	0.277778
<i>fx</i> = -0.081700 and <i>fxx</i> = 0.046800		

5.7.2 Derivatives of Equally Spaced Data

The first and second derivatives of a Newton forward-difference polynomial are given by Eqs. (5.26) and (5.27), respectively:

$$P'_n(x_0) = \frac{1}{h} \left(\Delta f_0 - \frac{1}{2} \Delta^2 f_0 + \frac{1}{3} \Delta^3 f_0 + \dots \right) \quad (5.124a)$$

$$P''_n(x_0) = \frac{1}{h^2} (\Delta^2 f_0 - \Delta^3 f_0 + \dots) \quad (5.124b)$$

The extrapolation formulas are given by Eqs. (5.116b) and (5.117).

$$\text{Error}(h/2) = \frac{1}{3}[f'(h/2) - f'(h)] \quad (5.125a)$$

$$\text{Extrapolated value} = f'(h/2) + \text{Error}(h/2) \quad (5.125b)$$

A FORTRAN subroutine, *subroutine deriv*, for evaluating Eqs (5.124) and (5.125) for $P_2(x)$ is presented in Program 5.4. It defines the data set and prints it, calls *subroutine deriv* to implement the solution, and prints the solution.

Program 5.4. Differentiation of a quadratic Newton forward-difference polynomial program.

```

program main
c   main program to illustrate numerical diff. subroutines
c   ndim  array dimension, n = 5 in this example
c   ndeg  degree of polynomial, ndeg = 2 in this example
c   num   number of derivative evaluations for extrapolation
c   n     number of data points
c   x     independent variable, x(i)
c   f     dependent variable, f(i)
c   fx    numerical approximation of derivative, fx(i,j)
dimension x(5),f(5),dx(2),fx(2,2),fxx(2,2)
data ndim,ndeg,num,n / 5, 2, 2, 5 /
data (x(i),i=1,5) / 3.3, 3.4, 3.5, 3.6, 3.7 /
data (f(i),i=1,5) / 0.303030, 0.294118, 0.285714, 0.277778,
1                           0.270270 /
write (6,1000)
do i=1,n
    write (6,1005) i,x(i),f(i)
end do
call deriv (ndim,ndeg,num,n,x,f,dx,fx,fxx)
write (6,1010)
write (6,1020) dx(1),fx(1,1),fx(1,2)
write (6,1030) dx(2),fx(2,1)
write (6,1050)
write (6,1040) dx(1),fxx(1,1),fxx(1,2)
write (6,1030) dx(2),fxx(2,1)
stop

```

```

1000 format (' Equally spaced data'// ' ' //      i',6x,'x',11x,'f'// ' ')
1005 format (i4,2f12.6)
1010 format (' /10x,'dx',8x,'0(h**2)',5x,'0(h**4)'// ' ')
1020 format (' fx ',3f12.6)
1030 format (4x,3f12.6)
1040 format (' fxx',3f12.6)
1050 format (' ')
end

subroutine deriv (ndim,ndeg,num,n,x,f,dx,fx,fxx)
c   numerical differentiation and extrapolation for P2(x)
dimension x(ndim),f(ndim),dx(num),fx(num,num),fxx(num,num)
dx(1)=x(3)-x(1)
dx(2)=x(2)-x(1)
fx(1,1)=0.5*(f(5)-f(1))/dx(1)
fx(2,1)=0.5*(f(4)-f(2))/dx(2)
fx(1,2)=(4.0*fx(2,1)-fx(1,1))/3.0
fxx(1,1)=(f(5)-2.0*f(3)+f(1))/dx(1)**2
fxx(2,1)=(f(4)-2.0*f(3)+f(2))/dx(2)**2
fxx(1,2)=(4.0*fxx(2,1)-fxx(1,1))/3.0
return
end

```

The data set used to illustrate *subroutine deriv* is taken from Examples 5.2 and 5.7. The output generated by the program is presented in Output 5.4.

Output 5.4. Solution by differentiation of a Newton forward-difference polynomial.

Equally spaced data

i	x	f
1	3.300000	0.303030
2	3.400000	0.294118
3	3.500000	0.285714
4	3.600000	0.277778
5	3.700000	0.270270
	dx	0(h**2) 0(h**4)
fx	0.200000	-0.081900 -0.081633
	0.100000	-0.081700
fxx	0.200000	0.046800 0.046800
	0.100000	0.046800

5.7.3. Packages for Numerical Differentiation

Numerous libraries and software packages are available for numerical differentiation. Many workstations and mainframe computers have such libraries attached to their operating systems.

Many commercial software packages contain numerical differentiation algorithms. Some of the more prominent packages are Matlab and Mathcad. More sophisticated packages, such as IMSL, MATHEMATICA, MACSYMA, and MAPLE, also contain routines for numerical differentiation. Finally, the book *Numerical Recipes* (Press et al., 1989) contains a routine for numerical differentiation.

5.8 SUMMARY

Procedures for numerical differentiation of discrete data and procedures for developing difference formulas are presented in this chapter. The numerical differentiation formulas are based on approximating polynomials. The direct fit polynomial, the Lagrange polynomial, and the divided difference polynomial work well for both unequally spaced data and equally spaced data. The Newton polynomials yield simple differentiation formulas for equally spaced data. Least squares fit polynomials can be used for large sets of data or sets of rough data.

Difference formulas, which approximate derivatives in terms of function values in the neighborhood of a particular point, are derived by both the Newton polynomial approach and the Taylor series approach. Difference formulas are used extensively in the numerical solution of differential equations.

After studying Chapter 5, you should be able to:

1. Describe the general features of numerical differentiation
2. Explain the procedure for numerical differentiation using direct fit polynomials
3. Apply direct fit polynomials to evaluate a derivative in a set of tabular data
4. Apply Lagrange polynomials to evaluate a derivative in a set of tabular data
5. Apply divided difference polynomials to evaluate a derivative in a set of tabular data
6. Describe the procedure for numerical differentiation using Newton forward-difference polynomials
7. Describe the procedure for numerical differentiation using Newton backward-difference polynomials
8. Describe the procedure for developing difference formulas from Newton difference polynomials
9. Develop a difference formula of any order for any derivative from Newton polynomials
10. Describe the procedure for developing difference formulas from Taylor series
11. Develop a difference formula of any order for any derivative by the Taylor series approach
12. Be able to use the difference formulas presented in Table 5.1
13. Explain the concepts of error estimation and extrapolation
14. Apply error estimation
15. Apply extrapolation

EXERCISE PROBLEMS

Table 1 gives values of $f(x) = \exp(x)$. This table is used in several of the problems in this chapter.

Table 1. Values of $f(x)$

x	$f(x)$	x	$f(x)$	x	$f(x)$
0.94	2.55998142	0.99	2.69123447	1.03	2.80106584
0.95	2.58570966	1.00	2.71828183	1.04	2.82921701
0.96	2.61169647	1.01	2.74560102	1.05	2.85765112
0.97	2.63794446	1.02	2.77319476	1.06	2.88637099
0.98	2.66445624				

5.2 Unequally Spaced Data

Direct Fit Polynomials

- For the data in Table 1, evaluate $f'(1.0)$ and $f''(1.0)$ using direct fit polynomials with the following data points: (a) 1.00 and 1.01, (b) 1.00, 1.01, and 1.02, and (c) 1.00, 1.01, 1.02, and 1.03. Compute and compare the errors.
- For the data in Table 1, evaluate $f'(1.0)$ and $f''(1.0)$ using direct fit polynomials with the following data points: (a) 0.99 and 1.00, (b) 0.98, 0.99, and 1.00, and (c) 0.97, 0.98, 0.99, and 1.00. Compute and compare the errors.
- For the data in Table 1, evaluate $f'(1.0)$ and $f''(1.0)$ using direct fit polynomials with the following data points: (a) 0.99 and 1.01, and (b) 0.99, 1.00, and 1.01. Compute and compare the errors.
- Compare the errors in Problems 1 to 3 and discuss.
- For the data in Table 1, evaluate $f'(1.0)$ and $f''(1.0)$ using direct fit polynomials with the following data points: (a) 0.98 and 1.02, (b) 0.98, 1.00, and 1.02, (c) 0.96 and 1.04, and (d) 0.96, 1.00, and 1.04. Compute the errors and compare the ratios of the errors for parts (a) and (c) and parts (b) and (d). Compare the results with the results of Problem 3.

Difference formulas can be derived from direct fit polynomials by fitting a polynomial to a set of symbolic data and differentiating the resulting polynomial. The truncation errors of such difference formulas can be obtained by substituting Taylor series into the difference formulas to recover the derivative being approximated accompanied by all of the neglected terms in the approximation. Use the symbolic Table 2, where Δx is considered constant, to work the following problems. Note that the algebra is simplified considerably by letting the base point value of x be zero and the other values of x be multiples of the constant increment size Δx .

Table 2. Symbolic Values of $f(x)$

x	$f(x)$	x	$f(x)$
x_{i-2}	f_{i-2}	x_{i+1}	f_{i+1}
x_{i-1}	f_{i-1}	x_{i+2}	f_{i+2}
x_i	f_i		

- Derive difference formulas for $f'(x)$ by direct polynomial fit using the following data points: (a) i and $i + 1$, (b) $i - 1$ and i , (c) $i - 1$ and $i + 1$, (d) $i - 1$, i , and $i + 1$, (e) i , $i + 1$, and $i + 2$, and (f) $i - 2$, $i - 1$, i , $i + 1$, and $i + 2$.

For each result, determine the leading truncation error term. Compare with the results presented in Table 5.1.

7. Derive difference formulas for $f''(x)$ by direct polynomial fit using the following data points: (a) $i - 1, i$, and $i + 1$, (b) $i, i + 1$, and $i + 2$, (c) $i - 2, i - 1, i, i + 1$, and $i + 2$, and (d) $i, i + 1, i + 2$, and $i + 3$. For each result, determine the leading truncation error term. Compare with the results presented in Table 5.1.

Lagrange Polynomials

8. Work Problem 1 using Lagrange polynomials.
9. Work Problem 2 using Lagrange polynomials.
10. Work Problem 3 using Lagrange polynomials.
11. Work Problem 5 using Lagrange polynomials.
12. Derive differentiation formulas for the third-degree Lagrange polynomial.

Divided Difference Polynomials

13. Work Problem 1 using divided difference polynomials.
14. Work Problem 2 using divided difference polynomials.
15. Work Problem 3 using divided difference polynomials.
16. Work Problem 5 using divided difference polynomials.

5.3 Equally Spaced Data

The data presented in Table 1 are used in the following problems. Construct a difference table for that set of data through third differences for use in these problems.

17. For the data in Table 1, evaluate $f'(1.0)$ and $f''(1.0)$ using Newton forward-difference polynomials of orders 1, 2, and 3 with the following points: (a) 1.00 to 1.03, and (b) 1.00 to 1.06. Compare the errors and ratios of the errors for the two increment sizes.
18. For the data in Table 1, evaluate $f'(1.0)$ and $f''(1.0)$ using Newton backward-difference polynomials of orders 1, 2, and 3 with the following points: (a) 0.97 to 1.00, and (b) 0.94 to 1.00. Compare the errors and the ratios of the errors for the two increment sizes.
19. For the data in Table 1, evaluate $f'(1.0)$ and $f''(1.0)$ using Newton forward-difference polynomials of orders 1 and 2 with the following points: (a) 0.99 to 1.01, and (b) 0.98 to 1.02. Compare the errors and the ratios of the errors for these two increment sizes. Compare with the results of Problems 17 and 18 and discuss.
20. Derive Eq. (5.23).
21. Derive Eq. (5.25).
22. Derive Eq. (5.40).
23. Derive Eq. (5.41).

Difference formulas can be derived from Newton polynomials by fitting a polynomial to a set of symbolic data and differentiating the resulting polynomial. The truncation error can be determined from the error term of the Newton polynomial. The symbolic data in Table 2 are used in the following problems. Construct a difference table for that set of data.

24. Derive difference formulas for $f'(x)$ using Newton forward-difference polynomials using the following data points: (a) i and $i + 1$, (b) $i - 1$ and i , (c) $i - 1$ and $i + 1$, (d) $i - 1, i$, and $i + 1$, (e) $i, i + 1$, and $i + 2$, and (f) $i - 2, i - 1, i, i + 1$, and $i + 2$. For each result, determine the leading truncation error term. Compare with the results presented in Table 5.1.
25. Derive difference formulas for $f''(x)$ using Newton forward-difference polynomials using the following data points: (a) $i - 1, i$, and $i + 1$, (b) $i, i + 1$, and $i + 2$, (c) $i - 2, i - 1, i, i + 1$, and $i + 2$, and (d) $i, i + 1, i + 2$, and $i + 3$. For each result, determine the leading truncation error term. Compare with the results presented in Table 5.1

5.4 Taylor Series Approach

26. Derive Eqs. (5.93) to (5.96).
27. Derive Eqs. (5.97) to (5.106).
28. Derive Eqs. (5.107) to (5.112).

5.5 Difference Formulas

29. Verify Eq. (5.96) by substituting Taylor series for the function values to recover the first derivative and the leading truncation error term.
30. Verify Eq. (5.99) by substituting Taylor series for the function values to recover the first derivative and the leading truncation error term.
31. Verify Eq. (5.109) by substituting Taylor series for the function values to recover the second derivative and the leading truncation error term.
32. Verify Eq. (5.92) by substituting Taylor series for the function values to recover the first derivative and the leading truncation error term.

5.6 Error Estimation and Extrapolation

33. For the data in Table 1, evaluate $f'(1.0)$ using Eq. (5.99) for $\Delta x = 0.04, 0.02$, and 0.01 . (a) Estimate the error for the $\Delta x = 0.02$ result. (b) Estimate the error for the $\Delta x = 0.01$ result. (c) Extrapolate the results to $O(\Delta x^4)$.
34. For the data in Table 1, evaluate $f''(1.0)$ using Eq. (5.109) for $\Delta x = 0.04, 0.02$, and 0.01 . (a) Estimate the error for the $\Delta x = 0.02$ result. (b) Estimate the error for the $\Delta x = 0.01$ result. (c) Extrapolate the results to $O(\Delta x^4)$.

5.7 Programs

35. Implement the program presented in Section 5.7.1 for the differentiation of a quadratic direct fit polynomial. Check out the program using the given data.
36. Solve Problems 1b and 2b using the program.
37. Modify the program to consider a linear direct fit polynomial. Solve Problems 1a and 2a using the program.
38. Extend the program to consider a cubic direct fit polynomial. Solve Problems 1c and 2c using the program.
39. Implement the program presented in Section 5.7.2 for the differentiation of a quadratic Lagrange polynomial. Check out the program using the given data.
40. Solve Problems 1b and 2b using the program.

41. Modify the program to consider a linear Lagrange polynomial. Solve Problems 1a and 2a using the program.
42. Extend the program to consider a cubic Lagrange polynomial. Solve Problems 1c and 2c using the program.
43. Implement the program presented in Section 5.7.3 for the differentiation of a quadratic divided difference polynomial. Check out the program using the given data.
44. Solve Problems 1b and 2b using the program.
45. Modify the program to consider a linear divided difference polynomial. Solve Problems 1a and 2a using the program.
46. Extend the program to consider a cubic divided difference polynomial. Solve Problems 1c and 2c using the program.
47. Implement the program presented in Section 5.7.4 for the differentiation of a quadratic Newton forward-difference polynomial. Check out the program using the given data.
48. Solve Problems 1b and 2b using the program.
49. Modify the program to consider a linear Newton forward-difference polynomial. Solve Problems 1a and 2a using the program.
50. Extend the program to consider a cubic Newton forward-difference polynomial. Solve Problems 1c and 2c using the program.

APPLIED PROBLEMS

51. When a fluid flows over a surface, the shear stress τ (N/m^2) at the surface is given by the expression

$$\tau = \mu \frac{du}{dy} \Big|_{\text{surface}} \quad (1)$$

where μ is the viscosity ($\text{N}\cdot\text{s}/\text{m}^2$), u is the velocity parallel to the surface (m/s), and y is the distance normal to the surface (cm). Measurements of the velocity of an air stream flowing above a surface are made with an LDV (laser-Doppler-velocimeter). The values given in Table 3 were obtained.

Table 3. Velocity Measurements

y	u	y	u
0.0	0.00	2.0	88.89
1.0	55.56	3.0	100.00

At the local temperature, $\mu = 0.00024 \text{ N}\cdot\text{s}/\text{m}^2$. Calculate (a) the difference table for $u(y)$, (b) du/dy at the surface based on first-, second-, and third-order polynomials, (c) the corresponding values of the shear stress at the surface, and (d) the shear force acting on a flat plate 10 cm long and 5 cm wide.

52. When a fluid flows over a surface, the heat transfer rate \dot{q} (J/s) to the surface is given by the expression

$$\dot{q} = -kA \frac{dT}{dy} \Big|_{\text{surface}} \quad (2)$$

where k is the thermal conductivity (J/s-m-K), T is the temperature (K), and y is the distance normal to the surface (cm). Measurements of the temperature of an air stream flowing above a surface are made with a thermocouple. The values given in Table 4 were obtained.

Table 4. Temperature Measurements

y	T	y	T
0.0	1000.00	2.0	355.56
1.0	533.33	3.0	300.00

At the average temperature, $k = 0.030$ J/s-m-K. Calculate (a) the difference table for $T(y)$, (b) dT/dy at the surface based on first-, second-, and third-order polynomials, (c) the corresponding values of the heat flux \dot{q}/A at the surface, and (d) the heat transfer to a flat plate 10 cm long and 5 cm wide.

6

Numerical Integration

- 6.1. Introduction
- 6.2. Direct Fit Polynomials
- 6.3. Newton-Cotes Formulas
- 6.4. Extrapolation and Romberg Integration
- 6.5. Adaptive Integration
- 6.6. Gaussian Quadrature
- 6.7. Multiple Integrals
- 6.8. Programs
- 6.9. Summary
- Problems

Examples

- 6.1. Direct fit polynomial
- 6.2. The trapezoid rule
- 6.3. Simpson's 1/3 rule
- 6.4. Romberg integration
- 6.5. Adaptive integration using the trapezoid rule
- 6.6. Gaussian quadrature
- 6.7. Double integral

6.1 INTRODUCTION

A set of tabular data is illustrated in Figure 6.1 in the form of a set of $[x, f(x)]$ pairs. The function $f(x)$ is known at a finite set of discrete values of x . Interpolation within a set of discrete data is presented in Chapter 4. Differentiation within a set of tabular data is presented in Chapter 5. *Integration* of a set of tabular data is presented in this chapter. The discrete data in Figure 6.1 are actually values of the function $f(x) = 1/x$, which is used as the example problem in this chapter.

The evaluation of integrals, a process known as *integration* or *quadrature*, is required in many problems in engineering and science.

$$I = \int_a^b f(x) dx$$

(6.1)

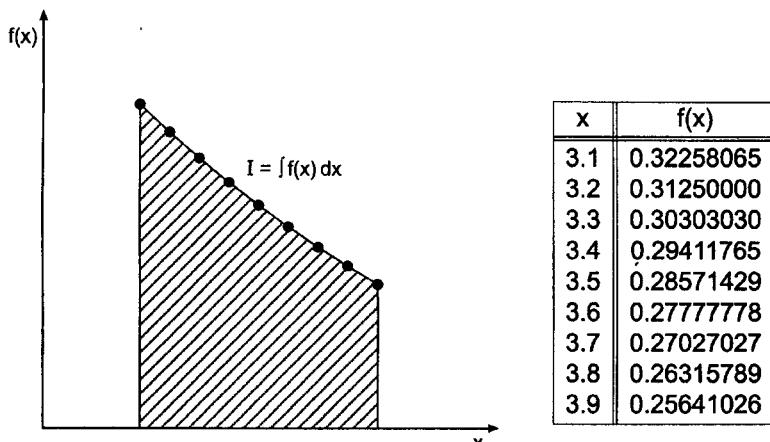


Figure 6.1 Integral of tabular data.

The function $f(x)$, which is to be integrated, may be a known function or a set of discrete data. Some known functions have an exact integral, in which case Eq. (6.1) can be evaluated exactly in closed form. Many known functions, however, do not have an exact integral, and an approximate numerical procedure is required to evaluate Eq. (6.1). In many cases, the function $f(x)$ is known only at a set of discrete points, in which case an approximate numerical procedure is again required to evaluate Eq. (6.1). The evaluation of integrals by approximate numerical procedures is the subject of this chapter.

Numerical integration (quadrature) formulas can be developed by fitting approximating functions (e.g., polynomials) to discrete data and integrating the approximating function:

$$I = \int_a^b f(x) dx \cong \int_a^b P_n(x) dx \quad (6.2)$$

This process is illustrated in Figure 6.2.

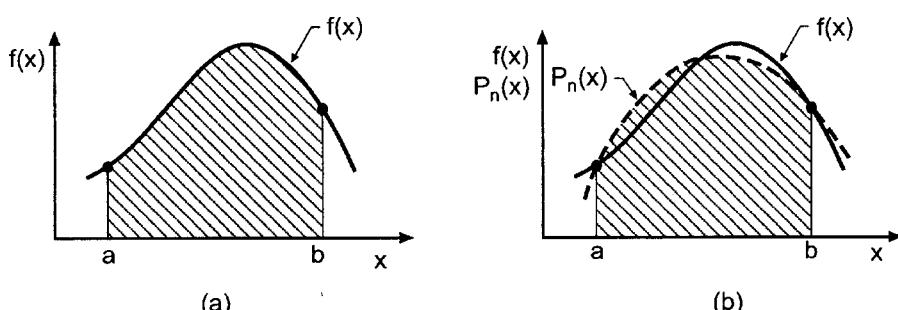


Figure 6.2 Numerical integration. (a) Exact integral. (b) Approximate integral.

Several types of problems arise. The function to be integrated may be known only at a finite set of discrete points. In that case, an approximating polynomial is fit to the discrete points, or several subsets of the discrete points, and the resulting polynomial, or polynomials, is integrated. The polynomial may be fit exactly to a set of points by the methods presented in Sections 4.3 to 4.6, or by a least squares fit as described in Section 4.10. In either case, the degree of the approximating polynomial chosen to represent the discrete data is the only parameter under our control.

When a known function is to be integrated, several parameters are under our control. The total number of discrete points can be chosen arbitrarily. The degree of the approximating polynomial chosen to represent the discrete data can be chosen. The locations of the points at which the known function is discretized can also be chosen to enhance the accuracy of the procedure.

Procedures are presented in this chapter for all of the situations discussed above. Direct fit polynomials are applied to prespecified unequally spaced data. Integration formulas based on Newton forward-difference polynomials, which are called *Newton-Cotes formulas*, are developed for equally spaced data. An important method, Romberg integration, based on extrapolation of solutions for successively halved increments, is presented. Adaptive integration, a procedure for minimizing the number of function evaluations required to integrate a known function, is discussed. Gaussian quadrature, which specifies the locations of the points at which known functions are discretized, is discussed. The numerical evaluation of multiple integrals is discussed briefly.

The simple function

$$f(x) = \frac{1}{x} \quad (6.3)$$

is considered in this chapter to illustrate numerical integration methods. In particular,

$$I = \int_{3.1}^{3.9} \frac{1}{x} dx = \ln(x) \Big|_{3.1}^{3.9} = \ln\left(\frac{3.9}{3.1}\right) = 0.22957444\dots \quad (6.4)$$

The procedures presented in this chapter for evaluating integrals lead directly into integration techniques for ordinary differential equations, which is discussed in Chapters 7 and 8.

The organization of Chapter 6 is illustrated in Figure 6.3. After the background material presented in this section, integration using direct fit polynomials as the approximating function is discussed. This section is followed by a development of Newton-Cotes formulas for equally spaced data, among which the trapezoid rule and Simpson's 1/3 rule are the most useful. Romberg integration, an extremely accurate and efficient algorithm which is based on extrapolation of the trapezoid rule, is presented next. Adaptive integration is then discussed. The following section presents Gaussian quadrature, an extremely accurate procedure for evaluating the integral of known functions in which the points at which the integral is evaluated is chosen in a manner which doubles the order of the integration formula. A brief introduction to multiple integrals follows. A summary closes the chapter and presents a list of what you should be able to do after studying Chapter 6.

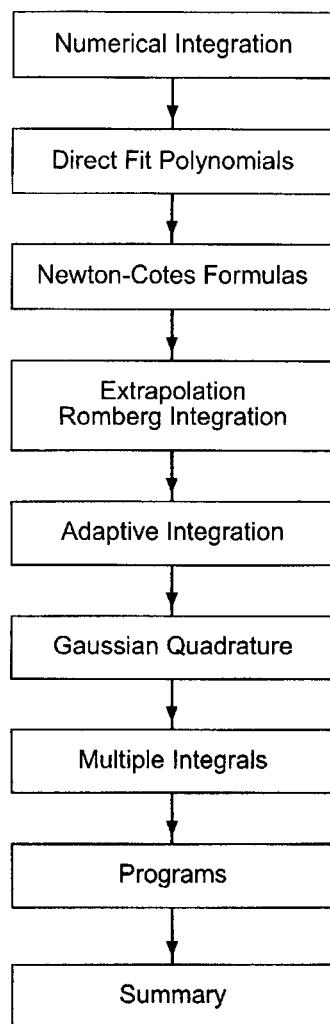


Figure 6.3 Organization of Chapter 6.

6.2 DIRECT FIT POLYNOMIALS

A straightforward numerical integration procedure that can be used for both unequally spaced data and equally spaced data is based on fitting the data by a direct fit polynomial and integrating that polynomial. Thus,

$$f(x) \cong P_n(x) = a_0 + a_1x + a_2x^2 + \dots \quad (6.5)$$

where $P_n(x)$ is determined by one of the following methods:

1. Given $N = n + 1$ sets of discrete data, $[x_i, f(x_i)]$, determine the exact n th-degree polynomial that passes through the data points, as discussed in Section 4.3.
2. Given $N > n + 1$ sets of discrete data, $[x_i, f(x_i)]$, determine the least squares n th-degree polynomial that best fits the data points, as discussed in Section 4.10.

3. Given a known function $f(x)$ evaluate $f(x)$ at N discrete points and fit a polynomial by an exact fit or a least squares fit.

After the approximating polynomial has been fit, the integral becomes

$$I = \int_a^b f(x) dx \cong \int_a^b P_n(x) dx \quad (6.6)$$

Substituting Eq. (6.5) into Eq. (6.6) and integrating yields

$$I = \left(a_0 x + a_1 \frac{x^2}{2} + a_2 \frac{x^3}{3} + \dots \right)_a^b \quad (6.7)$$

Introducing the limits of integration and evaluating Eq. (6.7) gives the value of the integral.

Example 6.1. Direct fit polynomial

Let's solve the example problem presented in Section 6.1 by a direct fit polynomial. Recall:

$$I = \int_{3.1}^{3.9} \frac{1}{x} dx \cong \int_{3.1}^{3.9} P_n(x) dx \quad (6.8)$$

Consider the following three data points from Figure 6.1:

x	$f(x)$
3.1	0.32258065
3.5	0.28571429
3.9	0.25641026

Fit the quadratic polynomial, $P_2(x) = a_0 + a_1x + a_2x^2$, to the three data points by the direct fit method:

$$0.32258065 = a_0 + a_1(3.1) + a_2(3.1)^2 \quad (6.9a)$$

$$0.28571429 = a_0 + a_1(3.5) + a_2(3.5)^2 \quad (6.9b)$$

$$0.25641026 = a_0 + a_1(3.9) + a_2(3.9)^2 \quad (6.9c)$$

Solving for a_0 , a_1 , and a_2 by Gauss elimination gives

$$P_2(x) = 0.86470519 - 0.24813896x + 0.02363228x^2 \quad (6.10)$$

Substituting Eq. (6.10) into Eq. (6.8) and integrating gives

$$I = [(0.86470519)x + \frac{1}{2}(-0.24813896)x^2 + \frac{1}{3}(0.02363228)x^3]_{3.1}^{3.9} \quad (6.11)$$

Evaluating Eq. (6.11) yields

$I = 0.22957974$

(6.12)

The error is Error = $0.22957974 - 0.22957444 = 0.00000530$.

6.3 NEWTON-COTES FORMULAS

The direct fit polynomial procedure presented in Section 6.2 requires a significant amount of effort in the evaluation of the polynomial coefficients. When the function to be integrated is known at equally spaced points, the Newton forward-difference polynomial presented in Section 4.6.2 can be fit to the discrete data with much less effort, thus significantly decreasing the amount of effort required. The resulting formulas are called *Newton-Cotes* formulas. Thus,

$$I = \int_a^b f(x) dx \cong \int_a^b P_n(x) dx \quad (6.13)$$

where $P_n(x)$ is the Newton forward-difference polynomial, Eq. (4.88):

$$\begin{aligned} P_n(x) = & f_0 + s \Delta f_0 + \frac{s(s-1)}{2} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{6} \Delta^3 f_0 \\ & + \dots + \frac{s(s-1)(s-2) \dots [s-(n-1)]}{n!} \Delta^n f_0 + \text{Error} \end{aligned} \quad (6.14)$$

where the interpolating parameter s is given by

$$s = \frac{x - x_0}{h} \rightarrow x = x_0 + sh \quad (6.15)$$

and the Error term is

$$\text{Error} = \binom{s}{n+1} h^{n+1} f^{(n+1)}(\xi) \quad x_0 \leq x \leq x_n \quad (6.16)$$

Equation (6.13) requires that the approximating polynomial be an explicit function of x , whereas Eq. (6.14) is implicit in x . Either Eq. (6.14) must be made explicit in x by introducing Eq. (6.16) into Eq. (6.14), or the second integral in Eq. (6.13) must be transformed into an explicit function of s , so that Eq. (6.14) can be used directly. The first approach leads to a complicated result, so the second approach is taken. Thus,

$$I = \int_a^b f(x) dx \cong \int_a^b P_n(x) dx = h \int_{s(a)}^{s(b)} P_n(s) ds \quad (6.17)$$

where, from Eq. (6.15)

$$dx = h ds \quad (6.18)$$

The limits of integration, $x = a$ and $x = b$, are expressed in terms of the interpolating parameter s by choosing $x = a$ as the base point of the polynomial, so that $x = a$ corresponds to $s = 0$ and $x = b$ corresponds to $s = s$. Introducing these results into Eq. (6.17) yields

$$I = h \int_0^s P_n(x_0 + sh) ds \quad (6.19)$$

Each choice of the degree n of the interpolating polynomial yields a different Newton-Cotes formula. Table 6.1 lists the more common formulas. Higher-order formulas have been developed [see Abramowitz and Stegun (1964)], but those presented in Table 6.1 are sufficient for most problems in engineering and science. The rectangle rule has

Table 6.1 Newton-Cotes Formulas

n	Formula
0	Rectangle rule
1	Trapezoid rule
2	Simpson's 1/3 rule
3	Simpson's 3/8 rule

poor accuracy, so it is not considered further. The other three rules are developed in this section.

Some terminology must be defined before proceeding with the development of the Newton-Cotes formulas. Figure 6.4 illustrates the region of integration. The distance between the lower and upper limits of integration is called the *range of integration*. The distance between any two data points is called an *increment*. A linear polynomial requires one *increment* and two data points to obtain a fit. A quadratic polynomial requires two increments and three data points to obtain a fit. And so on for higher-degree polynomials. The group of increments required to fit a polynomial is called an *interval*. A linear polynomial requires an interval consisting of only one increment. A quadratic polynomial requires an interval containing two increments. And so on. The total range of integration can consist of one or more intervals. Each interval consists of one or more increments, depending on the degree of the approximating polynomial.

6.3.1 The Trapezoid Rule

The trapezoid rule for a single interval is obtained by fitting a first-degree polynomial to two discrete points, as illustrated in Figure 6.5. The upper limit of integration x_1 corresponds to $s = 1$. Thus, Eq. (6.19) gives

$$\Delta I = h \int_0^1 (f_0 + s \Delta f_0) ds = h \left(sf_0 + \frac{s^2}{2} \Delta f_0 \right)_0^1 \quad (6.20)$$

where $h = \Delta x$. Evaluating Eq. (6.20) and introducing $\Delta f_0 = (f_1 - f_0)$ yields

$$\Delta I = h(f_0 + \frac{1}{2} \Delta f_0) = h[f_0 + \frac{1}{2}(f_1 - f_0)] \quad (6.21)$$

where ΔI denotes the integral for a single interval. Simplifying yields the trapezoid rule for a single interval:

$$\boxed{\Delta I = \frac{1}{2}h(f_0 + f_1)} \quad (6.22)$$

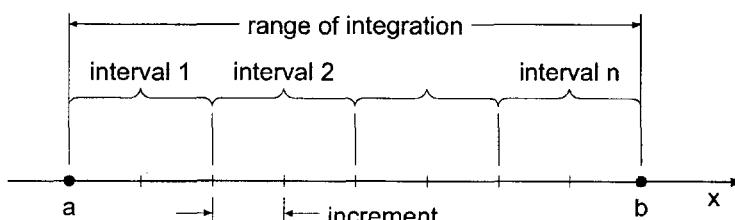


Figure 6.4 Range, intervals, and increments.

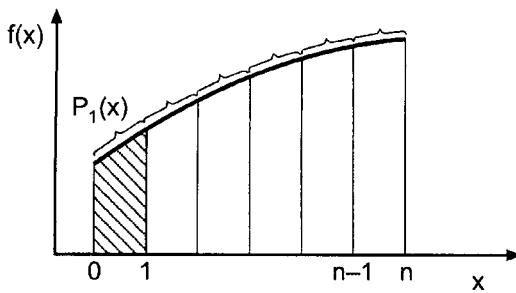


Figure 6.5 The trapezoid rule.

The composite trapezoid rule is obtained by applying Eq. (6.22) over all the intervals of interest. Thus,

$$I = \sum_{i=0}^{n-1} \Delta I_i = \sum_{i=0}^{n-1} \frac{1}{2} h_i (f_i + f_{i+1}) \quad (6.23)$$

where $h_i = (x_{i+1} - x_i)$. Equation (6.23) does not require equally spaced data. When the data are equally spaced, Eq. (6.23) simplifies to

$$I = \frac{1}{2} h (f_0 + 2f_1 + 2f_2 + \cdots + 2f_{n-1} + f_n) \quad (6.24)$$

where $\Delta x_i = \Delta x = h = \text{constant}$.

The error of the trapezoid rule for a single interval is obtained by integrating the error term given by Eq. (6.16). Thus,

$$\text{Error} = h \int_0^1 \frac{s(s-1)}{2} h^2 f''(\xi) ds = -\frac{1}{12} h^3 f''(\xi) = O(h^3) \quad (6.25)$$

Thus, the local error is $O(h^3)$. The total error for equally spaced data is given by

$$\sum_{i=0}^{n-1} \text{Error} = \sum_{i=0}^{n-1} -\frac{1}{12} h^3 f''(\xi) = n \left[-\frac{1}{12} h^3 f''(\bar{\xi}) \right] \quad (6.26)$$

where $x_0 \leq \bar{\xi} \leq x_n$. The number of steps $n = (x_n - x_0)/h$. Therefore,

$$\text{Total Error} = -\frac{1}{12} (x_n - x_0) h^2 f''(\bar{\xi}) = O(h^2) \quad (6.27)$$

Thus, the global (i.e., total) error is $O(h^2)$.

Example 6.2. The trapezoid rule

Let's solve the example problem presented in Section 6.1 by the trapezoid rule. Recall that $f(x) = 1/x$. Solving the problem for the range of integration consisting of only one interval of $h = 0.8$ gives

$$I(h = 0.8) = \frac{0.8}{2} (0.32258065 + 0.25641026) = 0.23159636 \quad (6.28)$$

Let's break the total range of integration into two intervals of $h = 0.4$ and apply the composite rule. Thus,

$$I(h = 0.4) = \frac{0.4}{2} [0.32258065 + 2(0.28571429) + 0.25641026] = 0.23008389 \quad (6.29)$$

For four intervals of $h = 0.2$, the composite rule yields

$$\begin{aligned} I(h = 0.2) &= \frac{0.2}{2} [0.32258065 + 2(0.30303030 + 0.28571429 \\ &\quad + 0.27027027) + 0.25641026] \\ &= 0.22970206 \end{aligned} \quad (6.30)$$

Finally, for eight intervals of $h = 0.1$,

$$\begin{aligned} I(h = 0.1) &= \frac{0.1}{2} [0.32258065 + 2(0.31250000 + \dots + 0.26315789) \\ &\quad + 0.25641026] \\ &= 0.22960636 \end{aligned} \quad (6.31)$$

Recall that the exact answer is $I = 0.22957444$.

The results are tabulated in Table 6.2, which also presents the errors and the ratios of the errors between successive interval sizes. The global error of the trapezoid rule is $O(h^2)$. Thus, for successive interval halvings,

$$\text{Ratio} = \frac{E(h)}{E(h/2)} = \frac{O(h^2)}{O(h/2)^2} = 2^2 = 4 \quad (6.32)$$

The results presented in Table 6.2 illustrate the second-order behavior of the trapezoid rule.

6.3.2 Simpson's 1/3 Rule

Simpson's 1/3 rule is obtained by fitting a second-degree polynomial to three equally spaced discrete points, as illustrated in Figure 6.6. The upper limit of integration x_2 corresponds to $s = 2$. Thus, Eq. (6.19) gives

$$\Delta I = h \int_0^2 \left[f_0 + s \Delta f_0 + \frac{s(s-1)}{2} \Delta^2 f_0 \right] ds \quad (6.33)$$

Table 6.2 Results for the Trapezoid Rule

h	I	Error	Ratio
0.8	0.23159636	-0.00202192	3.97
0.4	0.23008389	-0.00050945	3.99
0.2	0.22970206	-0.00012762	4.00
0.1	0.22960636	-0.00003192	

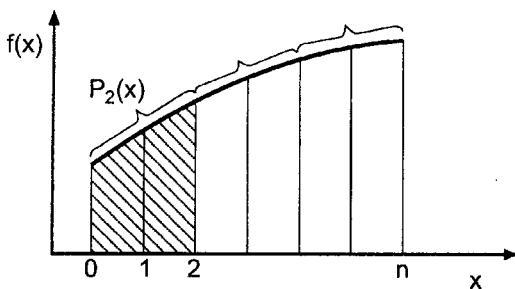


Figure 6.6 Simpson's 1/3 rule.

Performing the integration, evaluating the result, and introducing the expressions for Δf_0 and $\Delta^2 f_0$, yields Simpson's 1/3 rule for a single interval of two increments:

$$\Delta I = \frac{1}{3}h(f_0 + 4f_1 + f_2) \quad (6.34)$$

The composite Simpson's 1/3 rule for equally spaced points is obtained by applying Eq. (6.34) over the entire range of integration. Note that the total number of increments must be even. Thus,

$$I = \frac{1}{3}h(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 4f_{n-1} + f_n) \quad (6.35)$$

The error of Simpson's 1/3 rule for a single interval of two increments is obtained by evaluating the error term given by Eq. (6.16). Thus,

$$\text{Error} = h \int_0^2 \frac{s(s-1)(s-2)}{6} h^3 f'''(\xi) ds = 0 \quad (6.36)$$

This surprising result does not mean that the error is zero. It simply means that the cubic term is identically zero, and the error is obtained from the next term in the Newton forward-difference polynomial. Thus,

$$\text{Error} = h \int_0^2 \frac{s(s-1)(s-2)(s-3)}{24} h^4 f^{iv}(\xi) ds = -\frac{1}{90} h^5 f^{iv}(\xi) \quad (6.37)$$

Thus, the local error is $O(h^5)$. By an analysis similar to that performed for the trapezoid rule, it can be shown that the global error is $O(h^4)$.

Example 6.3. Simpson's 1/3 Rule

Let's solve the example problem presented in Section 6.1 using Simpson's 1/3 rule. Recall that $f(x) = 1/x$. Solving the problem for two increments of $h = 0.4$, the minimum permissible number of increments for Simpson's 1/3 rule, and one interval yields

$$I(h = 0.4) = \frac{0.4}{3} [0.32258065 + 4(0.28571429) + 0.25641026] = 0.22957974 \quad (6.38)$$

Table 6.3 Results for Simpson's 1/3 Rule

h	I	Error	Ratio
0.4	0.22957974	-0.000000530	
0.2	0.22967478	-0.000000034	15.59
0.1	0.22957446	-0.000000002	15.45

Breaking the total range of integration into four increments of $h = 0.2$ and two intervals and applying the composite rule yields:

$$\begin{aligned} I(h = 0.2) &= \frac{0.2}{3} [0.32258065 + 4(0.30303030) + 2(0.28571429) \\ &\quad + 4(0.27027027) + 0.25641026] \\ &= 0.22957478 \end{aligned} \quad (6.39)$$

Finally, for eight increments of $h = 0.1$ and four intervals,

$$\begin{aligned} I(h = 0.1) &= \frac{0.1}{3} [0.32258065 + 4(0.31250000) + 2(0.30303030) \\ &\quad + 4(0.29411765) + 2(0.28571429) + 4(0.27777778) \\ &\quad + 2(0.27027027) + 4(0.26315789) + 0.25641026] \\ &= 0.22957446 \end{aligned} \quad (6.40)$$

Recall that the exact answer is $I = 0.22957444$.

The results are tabulated in Table 6.3, which also presents the errors and the ratios of the errors between successive increment sizes. The global error of Simpson's 1/3 rule is $O(h^4)$. Thus, for successive increment halvings,

$$\text{Ratio} = \frac{E(h)}{E(h/2)} = \frac{O(h)^4}{O(h/2)^4} = 2^4 = 16 \quad (6.41)$$

The results presented in Table 6.3 illustrate the fourth-order behavior of Simpson's 1/3 rule.

6.3.3 Simpson's 3/8 Rule

Simpson's 3/8 rule is obtained by fitting a third-degree polynomial to four equally spaced discrete points, as illustrated in Figure 6.7. The upper limit of integration x_3 corresponds to $s = 3$. Thus, Eq. (6.19) gives

$$\Delta I = h \int_0^3 \left[f_0 + s \Delta f_0 + \frac{s(s-1)}{2} \Delta^2 f_0 + \frac{s(s-1)(s-2)}{6} \Delta^3 f_0 \right] ds \quad (6.42)$$

Performing the integration, evaluating the result, and introducing expressions for Δf_0 , $\Delta^2 f_0$, and $\Delta^3 f_0$ yields Simpson's 3/8 rule for a single interval of three increments:

$$\boxed{\Delta I = \frac{3}{8} h(f_0 + 3f_1 + 3f_2 + f_3)}$$

(6.43)

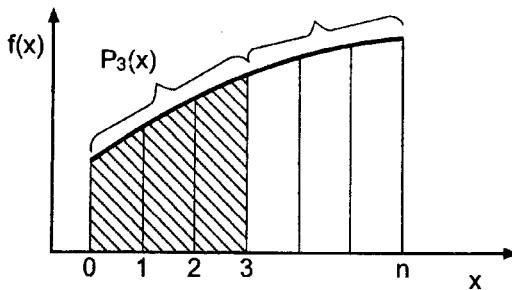


Figure 6.7 Simpson's 3/8 rule.

The composite Simpson's 3/8 rule for equally spaced points is obtained by applying Eq. (6.43) over the entire range of integration. Note that the total number of increments must be a multiple of three. Thus,

$$I = \frac{3}{8}h(f_0 + 3f_1 + 3f_2 + 2f_3 + 3f_4 + \cdots + 3f_{n-1} + f_n) \quad (6.44)$$

The error of Simpson's 3/8 rule for a single interval of three increments is obtained by evaluating the error term given by Eq. (6.16). Thus,

$$\text{Error} = h \int_0^3 \frac{s(s-1)(s-2)(s-3)}{24} h^4 f^{iv}(\xi) ds = -\frac{3}{80} h^5 f^{iv}(\xi) \quad (6.45)$$

Thus, the local error is $O(h^5)$ and the global error is $O(h^4)$.

Simpson's 1/3 rule and Simpson's 3/8 rule have the same order, $O(h^4)$, as shown by Eqs. (6.37) and (6.45). The coefficient in the local error of Simpson's 1/3 rule is $-1/90$, whereas the corresponding coefficient for Simpson's 3/8 rule is $-3/80$. Consequently, Simpson's 1/3 rule should be more accurate than Simpson's 3/8 rule. In view of this result, what use, if any, is Simpson's 3/8 rule? Simpson's 3/8 rule is useful when the total number of increments is odd. Three increments can be evaluated by the 3/8 rule, and the remaining even number of increments can be evaluated by the 1/3 rule.

6.3.4 Higher-Order Newton-Cotes Formulas

Numerical integration formulas based on equally spaced increments are called Newton-Cotes formulas. The trapezoid rule and Simpson's 1/3 and 3/8 rules are the first three Newton-Cotes formulas. The first 10 Newton-Cotes formulas are presented in Abramowitz and Stegun (1964). Newton-Cotes formulas can be expressed in the general form:

$$I = \int_a^b f(x) dx = n\beta h(\alpha_0 f_0 + \alpha_1 f_1 + \cdots) + \text{Error} \quad (6.46)$$

where n denotes both the number of increments and the degree of the polynomial, β and α_i are coefficients, and Error denotes the local error term. Table 6.4 presents n , β , α_i , and Error for the first seven Newton-Cotes formulas.

Table 6.4 Newton-Cotes Formulas

n	β	α_0	α_1	α_2	α_3	α_4	α_5	α_6	α_7	Local Error
1	1/2	1	1							$-1/12f^{(2)}h^3$
2	1/6	1	4	1						$-1/90f^{(4)}h^5$
3	1/8	1	3	3	1					$-3/80f^{(4)}h^5$
4	1/90	7	32	12	32	7				$-8/945f^{(6)}h^7$
5	1/288	19	75	50	50	75	19			$-275/12096f^{(6)}h^7$
6	1/840	41	216	27	272	27	216	41		$-9/1400f^{(8)}h^9$
7	1/17280	751	3577	1323	2989	2989	1323	3577	751	$-8183/518400f^{(8)}h^9$

6.4 EXTRAPOLATION AND ROMBERG INTEGRATION

When the functional form of the error of a numerical algorithm is known, the error can be estimated by evaluating the algorithm for two different increment sizes, as discussed in Section 5.6. The error estimate can be used both for error control and extrapolation. Recall the error estimation formula, Eq. (5.116b), written for the process of numerical integration, that is, with $f(h) = I(h)$. Thus,

$$\text{Error}(h/R) = \frac{1}{R^n - 1} [I(h/R) - I(h)] \quad (6.47)$$

where R is the ratio of the increment sizes and n is the global order of the algorithm. The extrapolation formula is given by Eq. (5.117):

$$\text{Extrapolated value} = f(h/R) + \text{Error}(h/R) \quad (6.48)$$

When extrapolation is applied to numerical integration by the trapezoid rule, the result is called *Romberg integration*. Recall the composite trapezoid rule, Eq. (6.24):

$$I = \sum_{i=0}^{n-1} \Delta I_i = \frac{1}{2} h(f_0 + 2f_1 + 2f_2 + \dots + 2f_{n-1} + f_n) \quad (6.49)$$

It can be shown that the error of the composite trapezoid rule has the functional form

$$\text{Error} = C_1 h^2 + C_2 h^4 + C_3 h^6 + \dots \quad (6.50)$$

Thus, the basic algorithm is $O(h^2)$, so $n = 2$. The following error terms increase in order in increments of 2.

Let's apply the trapezoid rule for a succession of smaller and smaller increment sizes, where each successive increment size is one-half of the preceding increment size. Thus, $R = h/(h/2) = 2$. Applying the error estimation formula, Eq. (6.47), gives

$$\text{Error}(h/2) = \frac{1}{2^n - 1} [I(h/2) - I(h)] \quad (6.51)$$

For the trapezoid rule itself, $n = 2$, and Eq. (6.51) becomes

$$\text{Error}(h/2) = \frac{1}{3} [I(h/2) - I(h)] \quad (6.52)$$

Equation (6.52) can be used for error estimation and error control.

Applying the extrapolation formula, Eq. (6.48), for $R = 2$ gives

$$\text{Extrapolated value} = f(h/2) + \text{Error}(h/2) + O(h^4) + \dots \quad (6.53)$$

Equation (6.53) shows that the result obtained by extrapolating the $O(h^2)$ trapezoid rule is $O(h^4)$.

If two extrapolated $O(h^4)$ values are available, which requires three $O(h^2)$ trapezoid rule results, those two values can be extrapolated to obtain an $O(h^6)$ value by applying Eq. (6.47) with $n = 4$ to estimate the $O(h^4)$ error, and adding that error to the more accurate $O(h^4)$ value. Successively higher-order extrapolations can be performed until round-off error masks any further improvement. Each successive higher-order extrapolation begins with an additional application of the $O(h^2)$ trapezoid rule, which is then combined with the previously extrapolated values to obtain the next higher-order extrapolated result.

Example 6.4. Romberg integration

Let's apply extrapolation to the results obtained in Example 6.2, in which the trapezoid rule is used to solve the example problem presented in Section 6.1. Recall that the exact answer is $I = 0.22957444$. Substituting $I(h = 0.8)$ and $I(h = 0.4)$ from Table 6.2 into Eq. (6.52) gives

$$\text{Error}(h/2) = \frac{1}{3}(0.23008389 - 0.23159636) = -0.00050416 \quad (6.54)$$

Substituting this result into Eq. (6.53) gives

$$\text{Extrapolated value} = 0.23008389 + (-0.00050416) = 0.22957973 \quad (6.55)$$

Repeating the procedure for the $h = 0.4$ and $h = 0.2$ results gives $\text{Error} = -0.00012728$ and $\text{Extrapolated value} = 0.22957478$. Both of the extrapolated values are $O(h^4)$. Substituting the two $O(h^4)$ extrapolated values into Eq. (6.51), with $n = 4$, gives

$$\text{Error}(h/2) = \frac{1}{2^4 - 1}(0.22957478 - 0.22957973) = -0.00000033 \quad (6.56)$$

Substituting this result into Eq. (6.53) gives

$$\text{Extrapolated value} = 0.22957478 + (-0.00000033) = 0.22957445 \quad (6.57)$$

These results, and the results of one more application of the trapezoid rule and its associated extrapolations, are presented in Table 6.5.

The $O(h^4)$ results are identical to the results for the $O(h^4)$ Simpson's 1/3 rule presented in Table 6.3. The second $O(h^6)$ result agrees with the exact value to eight significant digits.

Table 6.5 Romberg Integration

h	$I, O(h^2)$	Error	$O(h^4)$	Error	$O(h^6)$
0.8	0.23159636				
0.4	0.23008389	-0.00050416	0.22957973		
0.2	0.22970206	-0.00012728	0.22957478	-0.00000033	0.22957445
0.1	0.22960636	-0.00003190	0.22957446	-0.00000002	0.22957444

The results presented in Table 6.5 illustrate the error estimation and extrapolation concepts. Error control is accomplished by comparing the estimated error to a specified error limit and terminating the process when the comparison is satisfied. For example, if an error limit of $|0.00000100|$ is specified, that limit is obtained for the first $O(h^4)$ error estimate presented in Table 6.5. One would continue and add the error estimate of -0.00000033 to the $O(h^4)$ value to obtain the first $O(h^6)$ value, 0.22957445 , but the process would then be terminated and no further trapezoid rule values or extrapolations would be calculated. For the present case, the actual error for the more accurate $O(h^4)$ value is $\text{Error} = 0.22957478 - 0.22957444 = 0.00000034$, which is very close to the error estimate (except the signs are reversed as discussed below). The actual error in the corresponding $O(h^6)$ extrapolation is $\text{Error} = 0.22957445 - 0.22957444 = 0.00000001$.

The Error calculated by the extrapolation formula is based on the formula

$$f_{\text{exact}} = f(h) + \text{Error}(h)$$

which yields

$$\text{Error}(h) = f_{\text{exact}} - f(h)$$

The Error in the numerical results throughout this book is defined as

$$\text{Error}(h) = f(h) - f_{\text{exact}}$$

These two Error terms have the same magnitude, but opposite signs. Care must be exercised when both types of Error terms are discussed at the same time.

6.5 ADAPTIVE INTEGRATION

Any desired accuracy (within round-off limits) can be obtained by the numerical integration formulas presented in Section 6.3 by taking smaller and smaller increments. This approach is generally undesirable, since evaluation of the integrand function $f(x)$ is the most time-consuming portion of the calculation.

When the function to be integrated is known so that it can be evaluated at any location, the step size h can be chosen arbitrarily, so the increment can be reduced as far as desired. However, it is not obvious how to choose h to achieve a desired accuracy. Error estimation, as described in Section 6.4, can be used to choose h to satisfy a prespecified error criterion. Successive extrapolation, that is, Romberg integration, can be used to increase the accuracy further. This procedure requires the step size h to be a constant over the entire region of integration. However, the behavior of the integrand function $f(x)$ may not require a uniform step size to achieve the desired overall accuracy. In regions where the integrand function is varying slowly, only a few points should be required to achieve the desired accuracy. In regions where the integrand function is varying rapidly, a large number of points may be required to achieve the desired accuracy.

Consider the integrand function illustrated in Figure 6.8. In Region d–e, $f(x)$ is essentially constant, and the increment h may be very large. However, in region a–d, $f(x)$ varies rapidly, and the increment h must be very small. In fact, region a–d should be broken into three regions, as illustrated. Visual inspection of the integrand function can identify regions where h can be large or small. However, constructing a plot of the integrand function is time consuming and undesirable. A more straightforward automatic numerical procedure is required to break the overall region of integration into subregions in which the values of h may differ greatly.

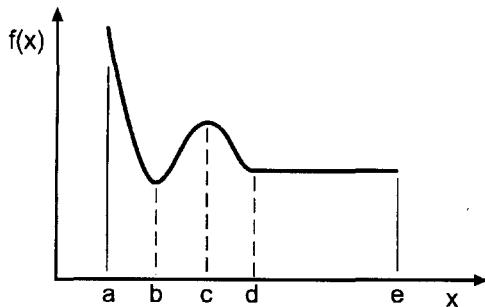


Figure 6.8 Integrand function.

Adaptive integration is a generic name denoting a strategy to achieve the desired accuracy with the minimum number of integrand function evaluations. A basic integration formula must be chosen, for example, the trapezoid rule or Simpson's 1/3 rule. The overall range of integration is broken into several subranges, and each subrange is evaluated to the desired accuracy by subdividing each individual subrange as required until the desired accuracy is obtained. Extrapolation may or may not be used in each subrange.

Example 6.5. Adaptive integration using the trapezoid rule

Let's illustrate adaptive integration by evaluating the following integral using the trapezoid rule:

$$I = \int_{0.1}^{0.9} \frac{1}{x} dx \quad (6.58)$$

The exact solution is

$$I = \ln(x) \Big|_{0.1}^{0.9} = \ln(0.9/0.1) = 2.197225 \dots \quad (6.59)$$

First, let's evaluate Eq. (6.58) with a uniform increment h over the total range of integration, starting with $h = (0.9 - 0.1) = 0.8$, and successively having h until the error estimate given by Eq. (6.52) is less than 0.001 in absolute value. Recall Eq. (6.52):

$$\text{Error}(h/2) = \frac{1}{3} [I(h/2) - I(h)] \quad (6.60)$$

The results are presented in Table 6.6. To satisfy the error criterion, $|\text{Error}| \leq 0.001$, 129 integrand function evaluations with $h = 0.00625$ are required. Extrapolating the final result yields

$$I = 2.197546 + (-0.000321) = 2.197225 \quad (6.61)$$

which agrees with the exact value to six digits after the decimal place.

Next, let's break the total range of integration into two subranges, $0.1 \leq x \leq 0.5$ and $0.5 \leq x \leq 0.9$, and apply the trapezoid rule in each subrange. The error criterion for the two subranges is $0.001/2 = 0.0005$. The results for the two subranges are presented in Table 6.7. To satisfy the error criterion requires 65 and 17 function evaluations,

Table 6.6 Integration Using the Trapezoid Rule

<i>n</i>	<i>h</i>	<i>I</i>	Error
2	0.80000	4.444444	
3	0.40000	3.022222	-0.474074
5	0.20000	2.463492	-0.186243
9	0.10000	2.273413	-0.063360
17	0.05000	2.217330	-0.018694
33	0.02500	2.202337	-0.004998
65	0.01250	2.198509	-0.001276
129	0.00625	2.197546	-0.000321

respectively, in the two subranges, for a total of 82 function evaluations. This is 47 less than before, which is a reduction of 36 percent. Extrapolating the two final results yields

$$I_1 = 1.609750 + (-0.000312) = 1.609438 \quad (6.62)$$

$$I_2 = 0.587931 + (-0.000144) = 0.587787 \quad (6.63)$$

which yields

$$I = I_1 + I_2 = 2.197225 \quad (6.64)$$

which agrees with the exact answer to six digits after the decimal place.

Example 6.5 is a simple example of adaptive integration. The extrapolation step increases the accuracy significantly. This suggests that using Romberg integration as the basic integration method within each subrange may yield a significant decrease in the number of function evaluations. Further increases in efficiency may be obtainable by subdividing the total range of integration into more than two subranges. More sophisti-

Table 6.7 Adaptive Integration Using the Trapezoid Rule

Subrange	<i>n</i>	<i>h</i>	<i>I</i>	Error
0.1 ≤ <i>x</i> ≤ 0.5	2	0.40000	2.400000	
	3	0.20000	1.866667	-0.177778
	5	0.10000	1.683333	-0.061111
	9	0.05000	1.628968	-0.018122
	17	0.02500	1.614406	-0.004854
	33	0.01250	1.610686	-0.001240
	65	0.00625	1.609750	-0.000312
0.5 ≤ <i>x</i> ≤ 0.9	2	0.40000	0.622222	
	3	0.20000	0.596825	-0.008466
	5	0.10000	0.590079	-0.002249
	9	0.05000	0.588362	-0.000572
	17	0.02500	0.587931	-0.000144

cated strategies can be employed to increase the efficiency of adaptive integration even further. The strategy employed in Example 6.5 is the simplest possible strategy.

6.6 GAUSSIAN QUADRATURE

The numerical integration methods presented in Section 6.3 are all based on equally spaced data. Consequently, if n points are considered, an $(n + 1)$ st-degree polynomial can be fit to the n points and integrated. The resulting formulas have the form:

$$I = \int_a^b f(x) dx = \sum_{i=1}^n C_i f(x_i) \quad (6.65)$$

where x_i are the locations at which the integrand function $f(x)$ is known and C_i are weighting factors. When the locations x_i are prespecified, this approach yields the best possible result. However, when a known function is to be integrated, an additional degree of freedom exists: the locations x_i at which the integrand function $f(x)$ is evaluated. Thus, if n points are used, $2n$ parameters are available: x_i ($i = 1, 2, \dots, n$) and C_i ($i = 1, 2, \dots, n$). With $2n$ parameters it is possible to fit a polynomial of degree $2n - 1$. Consequently, it should be possible to obtain numerical integration methods of much greater accuracy by choosing the values of x_i appropriately. *Gaussian quadrature* is one such method.

Gaussian quadrature formulas are obtained by choosing the values of x_i and C_i in Eq. (6.65) so that the integral of a polynomial of degree $2n - 1$ is exact. To simplify the development of the formulas, consider the integral of the function $F(t)$ between the limits of $t = -1$ and $t = +1$:

$$I = \int_{-1}^1 F(t) dt = \sum_{i=1}^n C_i F(t_i) \quad (6.66)$$

First, consider two points (i.e., $n = 2$), as illustrated in Figure 6.9. Choose t_1 , t_2 , C_1 , and C_2 so that I is exact for the following four polynomials: $F(t) = 1$, t , t^2 , and t^3 . Thus,

$$I[F(t) = 1] = \int_{-1}^1 (1) dt = t \Big|_{-1}^1 = 2 = C_1(1) + C_2(1) = C_1 + C_2 \quad (6.67a)$$

$$I[F(t) = t] = \int_{-1}^1 t dt = \frac{1}{2}t^2 \Big|_{-1}^1 = 0 = C_1 t_1 + C_2 t_2 \quad (6.67b)$$

$$I[F(t) = t^2] = \int_{-1}^1 t^2 dt = \frac{1}{3}t^3 \Big|_{-1}^1 = \frac{2}{3} = C_1 t_1^2 + C_2 t_2^2 \quad (6.67c)$$

$$I[F(t) = t^3] = \int_{-1}^1 t^3 dt = \frac{1}{4}t^4 \Big|_{-1}^1 = 0 = C_1 t_1^3 + C_2 t_2^3 \quad (6.67d)$$

Solving Eqs. (6.67) yields

$$C_1 = C_2 = 1 \quad t_1 = -\frac{1}{\sqrt{3}} \quad t_2 = \frac{1}{\sqrt{3}}$$

(6.68)

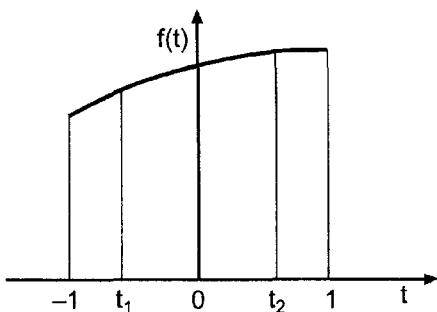


Figure 6.9 Gaussian quadrature.

Thus, Eq. (6.66) yields

$$I = \int_{-1}^1 F(t) dt = F\left(-\frac{1}{\sqrt{3}}\right) + F\left(\frac{1}{\sqrt{3}}\right) \quad (6.69)$$

The actual problem of interest is

$$I = \int_a^b f(x) dx \quad (6.70)$$

The problem presented in Eq. (6.70) can be transformed from x space to t space by the transformation

$$x = mt + c \quad (6.71)$$

where $x = a \rightarrow t = -1$, $x = b \rightarrow t = 1$, and $dx = m dt$. Thus,

$$a = m(-1) + c \quad \text{and} \quad b = m(1) + c \quad (6.72)$$

which gives

$$m = \frac{b-a}{2} \quad \text{and} \quad c = \frac{b+a}{2} \quad (6.73)$$

Thus, Eq. (6.71) becomes

$$x = \left(\frac{b-a}{2}\right)t + \frac{b+a}{2} \quad (6.74)$$

and Eq. (6.70) becomes

$$I = \int_a^b f(x) dx = \int_{-1}^1 f[x(t)] dt = \int_{-1}^1 f(mt + c)m dt \quad (6.75)$$

Define the function $F(t)$:

$$F(t) = f[x(t)] = f(mt + c) \quad (6.76)$$

Table 6.8 Gaussian Quadrature Parameters

<i>n</i>	<i>t_i</i>	<i>C_i</i>	Order
2	-1/ $\sqrt{3}$	1	3
	1/ $\sqrt{3}$	1	
3	- $\sqrt{0.6}$	5/9	5
	0	8/9	
	$\sqrt{0.6}$	5/9	
4	-0.8611363116	0.3478548451	7
	-0.3399810436	0.6521451549	
	0.3399810436	0.6521451549	
	0.8611363116	0.3478548451	

Substituting Eqs. (6.73) and (6.76) into Eq. (6.75) yields

$$I = \frac{b-a}{2} \int_{-1}^1 F(t) dt \quad (6.77)$$

Higher-order formulas can be developed in a similar manner. Thus,

$$\int_a^b f(x) dx = \frac{b-a}{2} \sum_{i=1}^n C_i F(t_i) \quad (6.78)$$

Table 6.8 presents t_i and C_i for $n = 2, 3$, and 4 . Higher-order results are presented by Abramowitz and Stegun (1964).

Example 6.6. Gaussian quadrature

To illustrate Gaussian quadrature, let's solve the example problem presented in Section 6.1, where $f(x) = 1/x$, $a = 3.1$, and $b = 3.9$. Consider the two-point formula applied to the total range of integration as a single interval. From Eq. (6.73),

$$m = \frac{b-a}{2} = 0.4 \quad \text{and} \quad c = \frac{b+a}{2} = 3.5 \quad (6.79)$$

Equations (6.74) and (6.76) become

$$x = 0.4t + 3.5 \quad \text{and} \quad F(t) = \frac{1}{0.4t + 3.5} \quad (6.80)$$

Substituting these results into Eq. (6.78) with $n = 2$ gives

$$I = 0.4 \int_{-1}^1 F(t) dt = 0.4 \left[(1)F\left(\frac{-1}{\sqrt{3}}\right) + (1)F\left(\frac{1}{\sqrt{3}}\right) \right] \quad (6.81)$$

Evaluating $F(t)$ gives

$$F\left(-\frac{1}{\sqrt{3}}\right) = \frac{1}{0.4(-1/\sqrt{3}) + 3.5} = 0.30589834 \quad (6.82a)$$

$$F\left(\frac{1}{\sqrt{3}}\right) = \frac{1}{0.4(1/\sqrt{3}) + 3.5} = 0.26802896 \quad (6.82b)$$

Substituting Eq. (6.82) into Eq. (6.81) yields

$$I = 0.4[(1)(0.30589834) + (1)(0.26802896)] = 0.22957092 \quad (6.83)$$

Recall that the exact value is $I = 0.22957444$. The error is $\text{Error} = 0.22957092 - 0.22957444 = -0.00000352$. This result is comparable to Simpson's 1/3 rule applied over the entire range of integration in a single step, that is, $h = 0.4$.

Next, let's apply the two-point formula over two intervals, each one-half of the total range of integration. Thus,

$$I = \int_{3.1}^{3.9} \frac{1}{x} dx = \int_{3.1}^{3.5} \frac{1}{x} dx + \int_{3.5}^{3.9} \frac{1}{x} dx = I_1 + I_2 \quad (6.84)$$

For I_1 , $a = 3.1$, $b = 3.5$, $(b - a)/2 = 0.2$, and $(b + a)/2 = 3.3$. Thus,

$$x = 0.2t + 3.3 \quad F(t) = \frac{1}{0.2t + 3.3} \quad (6.85)$$

$$I_1 = 0.2 \int_{-1}^1 F(t) dt = 0.2 \left[(1)F\left(-\frac{1}{\sqrt{3}}\right) + (1)F\left(\frac{1}{\sqrt{3}}\right) \right] \quad (6.86)$$

$$I_1 = 0.2 \left[\frac{1}{0.2(-1/\sqrt{3}) + 3.3} + \frac{1}{0.2(1/\sqrt{3}) + 3.3} \right] = 0.12136071 \quad (6.87)$$

For I_2 , $a = 3.5$, $b = 3.9$, $(b - a)/2 = 0.2$, and $(b + a)/2 = 3.7$. Thus,

$$x = 0.2t + 3.7 \quad F(t) = \frac{1}{0.2t + 3.7} \quad (6.88)$$

$$I_2 = 0.2 \int_{-1}^1 F(t) dt = 0.2 \left[(1)F\left(-\frac{1}{\sqrt{3}}\right) + (1)F\left(\frac{1}{\sqrt{3}}\right) \right] \quad (6.89)$$

$$I_2 = 0.2 \left[\frac{1}{0.2(-1/\sqrt{3}) + 3.7} + \frac{1}{0.2(1/\sqrt{3}) + 3.7} \right] = 0.10821350 \quad (6.90)$$

Summing the results yields the value of the total integral:

$$I = I_1 + I_2 = 0.12136071 + 0.10821350 = 0.22957421 \quad (6.91)$$

The error is $\text{Error} = 0.22957421 - 0.22957444 = -0.00000023$. This result is comparable to Simpson's 1/3 rule with $h = 0.2$.

Next let's apply the three-point formula over the total range of integration as a single interval. Thus,

$$a = 3.1 \quad b = 3.9 \quad \frac{b-a}{2} = 0.4 \quad \text{and} \quad \frac{b+a}{2} = 3. \quad (6.92)$$

$$x = 0.4t + 3.5 \quad f(t) = \frac{1}{0.4t + 3.5}$$

$$I = 0.4[\frac{5}{9}F(-\sqrt{0.6}) + \frac{8}{9}F(0) + \frac{5}{9}F(\sqrt{0.6})] \quad (6.93)$$

$$\begin{aligned} I &= 0.4 \left[\frac{5}{9} \frac{1}{0.4(-\sqrt{0.6}) + 3.5} + \frac{8}{9} \frac{1}{0.4(0) + 3.5} + \frac{5}{9} \frac{1}{0.4(\sqrt{0.6}) + 3.5} \right] \\ &= 0.22957443 \end{aligned} \quad (6.94)$$

The error is $\text{Error} = 0.22957443 - 0.22957444 = -0.00000001$. This result is comparable to Simpson's 1/3 rule with $h = 0.1$.

As a final example, let's evaluate the integral using the sixth-order formula based on the fifth-degree Newton forward-difference polynomial. That formula is (see Table 6.4)

$$I = \frac{5h}{288}(19f_0 + 75f_1 + 50f_2 + 50f_3 + 75f_4 + 19f_5) + O(h^7) \quad (6.95)$$

For five equally spaced increments, $h = (3.9 - 3.1)/5 = 0.16$. Thus,

$$\begin{aligned} I &= \frac{5(0.16)}{288}[19(1/3.10) + 75(1/3.26) + 50(1/3.42) + 50(1/3.58) \\ &\quad + 75(1/3.74) + 19(1/3.90)] \\ &= 0.22957445 \end{aligned} \quad (6.96)$$

The error is $\text{Error} = 0.22957445 - 0.22957444 = 0.00000001$. This result is comparable to Gaussian quadrature with three points.

6.7 MULTIPLE INTEGRALS

The numerical integration formulas developed in the preceding sections for evaluating single integrals can be used to evaluate multiple integrals. Consider the double integral:

$$I = \int_c^d \int_a^b f(x, y) dx dy \quad (6.97)$$

Equation (6.97) can be written in the form:

$$I = \int_c^d \left(\int_a^b f(x, y) dx \right) dy = \int_c^d F(y) dy \quad (6.98)$$

where

$$F(y) = \int_a^b f(x, y) dx \quad y = \text{Constant} \quad (6.99)$$

The double integral is evaluated in two steps:

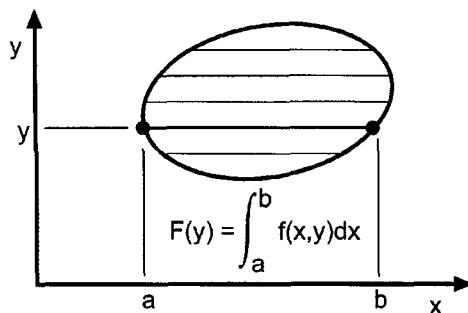


Figure 6.10 Double integration.

1. Evaluate $F(y)$ at selected values of y by any numerical integration formula.
2. Evaluate $I = \int F(y) dy$ by any numerical integration formula.

If the limits of integration are variable, as illustrated in Figure 6.10, that must be accounted for.

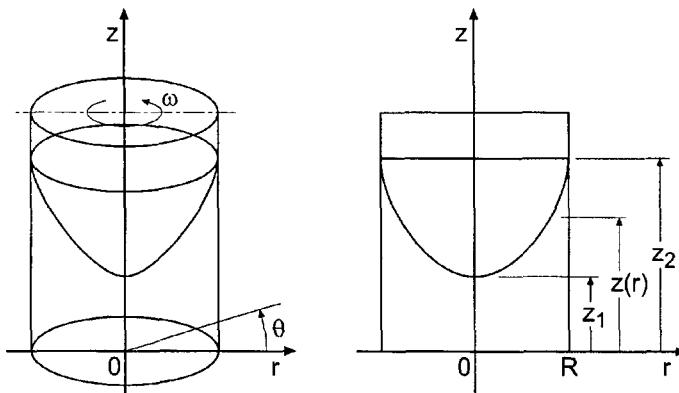
Example 6.7. Double integral

To illustrate double integration with variable limits of integration, let's calculate the mass of water in a cylindrical container which is rotating at a constant angular velocity ω , as illustrated in Figure 6.11a. A meridional plane view through the axis of rotation is presented in Figure 6.11b. From a basic fluid mechanics analysis, it can be shown that the shape of the free surface is given by

$$z(r) = A + Br^2 \quad (6.100)$$

From measured data, $z(0) = z_1$ and $z(R) = z_2$. Substituting these values into Eq. (6.100) gives

$$z(r) = z_1 + \frac{(z_2 - z_1)r^2}{R^2} \quad (6.101)$$



(a) Physical arrangement.

(b) Meridional plane view.

Figure 6.11 Spinning cylindrical container.

In a specific experiment, $R = 10.0 \text{ cm}$, $z_1 = 10.0 \text{ cm}$, and $z_2 = 20.0 \text{ cm}$. In this case, Eq. (6.101) gives

$$z(r) = 10.0 + 0.1r^2 \quad (6.102)$$

Let's calculate the mass of water in the container at this condition. The density of water is $\rho = 1.0 \text{ g/cm}^3$. Due to axial symmetry in the geometry of the container and the height distribution, the mass in the container can be expressed in cylindrical coordinates as

$$m = \int dm = \int_0^R \rho z(r) (2\pi r dr) = 2\pi\rho \int_0^R \left[z_1 + \frac{(z_2 - z_1)r^2}{R^2} \right] dr \quad (6.103)$$

which has the exact integral

$$m = \pi\rho \left[z_1 R^2 + \frac{(z_2 - z_1)R^2}{2} \right] \quad (6.104)$$

Substituting the specified values of ρ , R , z_1 , and z_2 into Eq. (6.104) yields $m = 1500\pi g = 4712.388980 \text{ g}$.

To illustrate the process of numerical double integration, let's solve this problem in Cartesian coordinates. Figure 6.12(a) illustrates a discretized Cartesian grid on the bottom of the container. In Cartesian coordinates, $dA = dx dy$, and the differential mass in a differential column of height $z(r)$ is given by

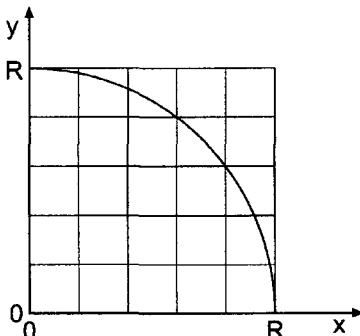
$$dm = \rho z(r) dA \quad (6.105)$$

Substituting Eq. (6.101) into Eq. (6.105), where $r^2 = x^2 + y^2$, and integrating gives

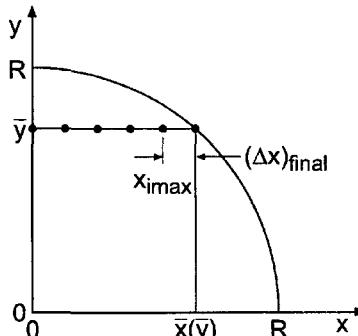
$$m = \int dm = \rho \iint [z_1 + (z_2 - z_1)(x^2 + y^2)R^{-2}] dx dy \quad (6.106)$$

Substituting the specific values of ρ , R , z_1 , and z_2 into Eq. (6.106) gives

$$m = 1.0 \iint [10 + 0.1(x^2 + y^2)] dx dy \quad (6.107)$$



(a) Discretized grid.



(b) Relationship of \bar{x} and \bar{y} .

Figure 6.12 Cartesian coordinates.

Table 6.9 Geometrical Parameters

<i>j</i>	\bar{y} , cm	$\bar{x}(\bar{y})$, cm	$i_{\max}(\bar{y})$	$x(i_{\max}(\bar{y}))$	$(\Delta x)_{\text{final}}$
1	0.0	10.000000	11	10.0	0.000000
2	1.0	9.949874	10	9.0	0.949874
3	2.0	9.797959	10	9.0	0.797959
4	3.0	9.539392	10	9.0	0.539392
5	4.0	9.165151	10	9.0	0.165151
6	5.0	8.660254	9	8.0	0.660254
7	6.0	8.000000	9	8.0	0.000000
8	7.0	7.141428	8	7.0	0.141428
9	8.0	6.000000	7	6.0	0.000000
10	9.0	4.358899	5	4.0	0.358899
11	10.0	0.000000	1	0.0	0.000000

Table 6.10 Integrand of Eq. (6.111) at $\bar{y} = 5.0$ cm

<i>i</i>	$F(x)$	<i>i</i>	$F(x)$
1	12.500	6	15.000
2	12.600	7	16.100
3	12.900	8	17.400
4	13.400	9	18.900
5	14.100	10	20.000

Due to symmetry about the x and y axes, Eq. (6.107) can be expressed as

$$m = 4(1.0) \int_0^R \left[\int_0^{\bar{x}(\bar{y})} [10 + 0.1(x^2 + \bar{y}^2)] dx \right] dy = 4 \int_0^R F(\bar{y}) dy \quad (6.108)$$

where $F(\bar{y})$ is defined as

$$F(\bar{y}) = \int_0^{\bar{x}(\bar{y})} [10.0 + 0.1(x^2 + \bar{y}^2)] dx \quad (6.109)$$

and \bar{y} and $\bar{x}(\bar{y})$ are illustrated in Figure 6.12(b). Thus,

$$\bar{x} = (R^2 - \bar{y}^2)^{1/2} \quad (6.110)$$

Table 6.11 Values of $F(\bar{y})$

<i>j</i>	\bar{y} , cm	$F(\bar{y})$	<i>j</i>	\bar{y} , cm	$F(\bar{y})$
1	0.0	133.500000	7	6.0	126.000000
2	1.0	133.492600	8	7.0	118.664426
3	2.0	133.410710	9	8.0	105.700000
4	3.0	133.068144	10	9.0	81.724144
5	4.0	132.128255	11	10.0	0.000000
6	5.0	130.041941			

Table 6.12. Results for $nx = 11, 21, 41, 81$ and 161

nx	m, g	Error, g	Error ratio
11	4643.920883	-68.468097	2.75
21	4687.527276	-24.861705	2.78
41	4703.442452	-8.946528	2.80
81	4709.188100	-3.200880	2.81
161	4711.248087	-1.140893	

Let's discretize the y axis into 10 equally spaced increments with $\Delta y = 1.0$ cm. For each value of $\bar{y} = (j - 1) \Delta y$ ($j = 1, 2, \dots, 11$) let's calculate $\bar{x}(\bar{y})$ from Eq. (6.110). At each value of \bar{y} , let's discretize the x axis into $(\text{imax}(\bar{y}) - 1)$ equally spaced increments with $\Delta x = 1.0$ cm, with a final increment $(\Delta x)_{\text{final}}$ between $x = (\text{imax}(\bar{y}) - 1) \Delta x$ and $\bar{x}(\bar{y})$. The resulting geometrical parameters are presented in Table 6.9.

The values of $F(\bar{y})$, defined in Eq. (6.109), are evaluated by the trapezoid rule. As an example, consider $j = 6$ for which $\bar{y} = 5.0$ cm. Equation (6.109) becomes

$$F(5.0) = \int_{0.0}^{8.000000} [10.0 + 0.1(x^2 + 25.0)] dx = \int_{0.0}^{8.000000} F(x) dx \quad (6.111)$$

Table 6.10 presents the integrand $F(x)$ of Eq. (6.111).

Integrating Eq. (6.111) by the trapezoid rule gives

$$\begin{aligned} F(5.0) &= \frac{1.0}{2} [12.500 + 2(12.600 + 12.900 + 13.400 + 14.000 + 15.000 \\ &\quad + 16.100 + 17.400) + 18.900] + \frac{0.660254}{2} (18.900 + 20.000) \\ &= 130.041941 \end{aligned} \quad (6.112)$$

Repeating this procedure for every value of \bar{y} in Table 6.9 yields the results presented in Table 6.11.

Integrating Eq. (6.108) by the trapezoid rule, using the values of $F(\bar{y})$ presented in Table 6.11, yields

$$\begin{aligned} m &= 4.0 \frac{1.0}{2} [133.500 + 2(133.492600 + 133.410710 + 133.068144 + 132.128255 \\ &\quad + 130.041941 + 126.000000 + 118.664426 + 105.700000 + 81.724144) \\ &\quad + 0.000000] \\ &= 4643.920883 \text{ g} \end{aligned} \quad (6.113)$$

The error is $\text{Error} = 4643.920883 - 4712.388980 = -68.468047$ g. Repeating the calculations for $nx = 21, 41, 81$, and 161 yields the results presented in Table 6.12. These results show the procedure is behaving better than first order, but not quite second order.

Example 6.7 illustrates the complexity of multiple integration by numerical methods, especially with variable limits of integration. The procedure is not difficult, just complicated. The procedure can be extended to three or more independent variables in a straightforward manner, with a corresponding increase in complexity.

Example 6.7 uses the trapezoid rule. Simpson's 1/3 rule could be used instead of the trapezoid rule. When the integrand is a known function, adaptive integration techniques can be employed, as can Gaussian quadrature.

6.8 PROGRAMS

Three FORTRAN subroutines for numerical integration are presented in this section:

1. The trapezoid rule
2. Simpson's 1/3 rule
3. Romberg integration

The basic computational algorithms are presented as completely self-contained subroutines suitable for use in other programs. Input data and output statements are contained in a main (or driver) program written specifically to illustrate the use of each subroutine.

6.8.1. The Trapezoid Rule

The general algorithm for the trapezoid rule is give by Eq. (6.24):

$$I = \frac{1}{2}h(f_0 + 2f_1 + 2f_2 + \cdots + 2f_{n-1} + f_n) \quad (6.114)$$

A FORTRAN subroutine, *subroutine trap*, for implementing the trapezoid rule is presented in Program 6.1. *Program main* defines the data set and prints it, calls *subroutine trap* to implement the solution, and prints the solution (Output 6.1).

Program 6.1 The trapezoid rule program.

```

program main
c   main program to illustrate numerical integration subroutines
c   ndim  array dimension, ndim = 9 in this example
c   n      number of data points
c   x      independent variable array, x(i)
c   f      dependent variable array, f(i)
c   sum    value of the integral
dimension x(9),f(9)
data ndim,n / 9, 9 /
data (x(i),i=1,9)/3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9/
data (f(i),i=1,9) / 0.32258065, 0.31250000, 0.30303030,
1 0.29411765, 0.28571429, 0.27777778, 0.27027027, 0.26315789,
2 0.25641026 /
write (6,1000)
do i=1,n
    write (6,1010) i,x(i),f(i)
end do
call trap (ndim,n,x,f,sum)
write (6,1020) sum
stop
1000 format (' Trapezoid rule'// ' '// i',7x,'x',13x,'f'// ' ')
1010 format (i4,2f14.8)
1020 format (' '// I '=',f14.8)
end

subroutine trap (ndim,n,x,f,sum)
c   trapezoid rule integration
dimension x(ndim),f(ndim)
sum=f(1)+f(n)

```

```

do i=2,n-1
    sum=sum+2.0*f(i)
end do
sum=sum*(x(n)-x(1))/float(n-1)/2.0
return
end

```

The data set used to illustrate *subroutine trap* is taken from Example 6.2. The output generated by the trapezoid rule program is presented in Output 6.1.

Output 6.1 Solution by the trapezoid rule

Trapezoid rule

i	x	f
1	3.10000000	0.32258065
2	3.20000000	0.31250000
3	3.30000000	0.30303030
4	3.40000000	0.29411765
5	3.50000000	0.28571429
6	3.60000000	0.27777778
7	3.70000000	0.27027027
8	3.80000000	0.26315789
9	3.90000000	0.25641026

I = 0.22960636

6.8.2. Simpson's 1/3 Rule

The general algorithm for Simpson's 1/3 rule is given by Eq. (6.35):

$$I = \frac{1}{3}h(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 4f_{n-1} + f_n) \quad (6.115)$$

A FORTRAN subroutine, *subroutine simpson*, for implementing Simpson's 1/3 rule is presented in Program 6.2. *Subroutine simpson* works essentially like *subroutine trap* discussed in Section 6.8.1, except Simpson's 1/3 rule is used instead of the trapezoid rule. *Program main* defines the data set and prints it, calls *subroutine simpson* to implement Simpson's 1/3 rule, and prints the solution. Only the statements in *program main* which are different from *program main* in Section 6.8.1 are presented.

Program 6.2 Simpson's 1/3 rule program

```

program main
c      main program to illustrate numerical integration subroutines
      call simpson (ndim,n,x,f,sum)
1000 format (' Simpsons 1/3 rule'// ' ' i',7x,'x',13x,'f'// ' ')
      end

      subroutine simpson (ndim,n,x,f,sum)

```

```

c      Simpson's 1/3 rule integration
      dimension x(ndim),f(ndim)
      sum2=0.0
      sum4=0.0
do i=3,n-1,2
  sum2=sum2+f(i)
end do
do i=2,n-1,2
  sum4=sum4+f(i)
end do
sum=(f(1)+2.0*sum2+4.0*sum4+f(n))*(x(2)-x(1))/3.0
return
end

```

The data set used to illustrate *subroutines simpson* is taken from Example 6.3. The output generated by Simpson's 1/3 rule program is presented in Output 6.2.

Output 6.2 Solution by Simpson's 1/3 rule

Simpsons 1/3 rule

i	x	f
1	3.10000000	0.32258065
2	3.20000000	0.31250000
3	3.30000000	0.30303030
4	3.40000000	0.29411765
5	3.50000000	0.28571429
6	3.60000000	0.27777778
7	3.70000000	0.27027027
8	3.80000000	0.26315789
9	3.90000000	0.25641026

I = 0.22957446

6.8.3. Romberg Integration

Romberg integration is based on extrapolation of the trapezoid rule, Eq. (6.114). The general extrapolation formula for Romberg integration is given by Eq. (6.51):

$$\text{Error}(h/2) = \frac{1}{2^n - 1} [I(h/2) - I(h)] \quad (6.116)$$

A FORTRAN subroutine, *subroutine romberg*, for implementing the procedure is presented in Program 6.3. *Program main* defines the data set and prints it, calls *subroutine romberg* to implement the solution, and prints the solution (Output 6.3). Only the statements in *program main* which are different from *program main* in Section 6.8.1 are presented.

Program 6.3 Romberg integration program

```

program main
c   main program to illustrate numerical integration subroutines
c   s      array of integrals and extrapolations, s(i,j)
dimension x(9),f(9),s(9,9)
data ndim,n,num / 9, 9, 4 /
call romberg (ndim,n,num,x,f,s)
write (6,1020)
do i=1,num
    write (6,1010) i,(s(i,j),j=1,num+1-i)
end do
stop
1000 format (' Romberg integration'// ' ' i',7x,'x',13x,'f'// ' ')
1010 format (i3,4f14.8)
1020 format (' '// i',6x,'s(i,1)',8x,'s(i,2)',8x,'s(i,3)',8x,
1 's(i,4)'// ' ')
end

subroutine romberg (ndim,n,num,x,f,s)
c   Romberg integration
dimension x(ndim),f(ndim),s(ndim,ndim)
c   trapezoid rule integration
k=n
dx=(x(n)-x(1))
s(1,1)=(f(1)+f(n))*dx/2.0
do j=2,num
    dx=dx/2.0
    sum=0.0
    k=k/2
    do i=k+1,n-1,k
        sum=sum+f(i)
    end do
    s(j,1)=(f(1)+2.0*sum+f(n))*dx/2.0
end do
c   Romberg extrapolation
do j=2,num
    ex=float(2*(j-1))
    den=2.0**ex-1.0
    k=num+1-j
    do i=1,k
        s(i,j)=s(i+1,j-1)+(s(i+1,j-1)-s(i,j-1))/den
    end do
end do
return
end

```

The data set used to illustrate *subroutine romberg* is taken from Example 6.5. The output is presented in Output 6.3.

Output 6.3 Solution by Romberg integration

Romberg integration

i	x	f		
1	3.1000000	0.32258065		
2	3.2000000	0.31250000		
3	3.3000000	0.30303030		
4	3.4000000	0.29411765		
5	3.5000000	0.28571429		
6	3.6000000	0.27777778		
7	3.7000000	0.27027027		
8	3.8000000	0.26315789		
9	3.9000000	0.25641026		
i	s(i,1)	s(i,2)	s(i,3)	s(i,4)
1	0.23159636	0.22957974	0.22957445	0.22957444
2	0.23008390	0.22957478	0.22957444	
3	0.22970206	0.22957446		
4	0.22960636			

6.8.4. Packages for Numerical Integration

Numerous libraries and software packages are available for numerical integration. Many workstations and mainframe computers have such libraries attached to their operating systems.

Many commercial software packages contain numerical integration algorithms. Some of the more prominent are Matlab and Mathcad. More sophisticated packages, such as IMSL, MATHEMATICA, MACSYMA, and MAPLE, also contain numerical integration algorithms. Finally, the book *Numerical Recipes* [Press et al. (1989)] contains numerous routines for numerical integration.

6.9 SUMMARY

Procedures for the numerical integration of both discrete data and known functions are presented in this chapter. These procedures are based on fitting approximating polynomials to the data and integrating the approximating polynomials. The direct fit polynomial method works well for both equally spaced data and nonequally spaced data. Least squares fit polynomials can be used for large sets of data or sets of rough data. The Newton-Cotes formulas, which are based on Newton forward-difference polynomials, give simple integration formulas for equally spaced data.

Methods of error estimation and error control are presented. Extrapolation, or the deferred approach to the limit, is discussed. Romberg integration, which is extrapolation of the trapezoid rule, is developed. Adaptive integration procedures are suggested to reduce the effort required to integrate widely varying functions. Gaussian quadrature, which

increases the accuracy of integrating known functions where the sampling locations can be chosen arbitrarily, is presented. An example of multiple integration is presented to illustrate the extension of the one-dimensional integration formulas to multiple dimensions.

Of all the methods considered, it is likely that Romberg integration is the most efficient. Simpson's rules are elegant and intellectually interesting, but the first extrapolation of Romberg integration gives comparable results. Subsequent extrapolations of Romberg integration increase the order at a fantastic rate. Simpson's 1/3 rule could be developed into an extrapolation procedure, but that yields no advantage over Romberg integration.

After studying Chapter 6, you should be able to:

1. Describe the general features of numerical integration
2. Explain the procedure for numerical integration using direct fit polynomials
3. Apply direct fit polynomials to integrate a set of tabular data
4. Describe the procedure for numerical integration using Newton forward-difference polynomials
5. Derive the trapezoid rule
6. Apply the trapezoid rule
7. Derive Simpson's 1/3 rule
8. Apply Simpson's 1/3 rule
9. Derive Simpson's 3/8 rule
10. Apply Simpson's 3/8 rule
11. Describe the relative advantage and disadvantages of Simpson's 1/3 rule and Simpson's 3/8 rule
12. Derive Newton-Cotes formulas of any order
13. Apply Newton-Cotes formulas of any order
14. Explain the concepts of error estimation, error control, and extrapolation
15. Describe Romberg integration
16. Apply Romberg integration
17. Describe the concept of adaptive integration
18. Develop and apply a simple adaptive integration algorithm
19. Explain the concepts underlying Gaussian quadrature
20. Derive the two-point Gaussian quadrature formula
21. Describe how to develop Gaussian quadrature formula for more than two points
22. Apply Gaussian quadrature
23. Describe the process for numerically evaluating multiple integrals
24. Develop a procedure for evaluating double integrals
25. Apply the double integral numerical integration procedure

EXERCISE PROBLEMS

6.1

The following integrals are used throughout this chapter to illustrate numerical integration methods. All of these integrals have exact solutions, which should be used for error analysis in the numerical problems.

- | | |
|---|---|
| (A) $\int_0^5 (3x^2 + 2) dx$ | (B) $\int_{1.0}^{2.0} (x^3 + 3x^2 + 2x + 1) dx$ |
| (C) $\int_0^{10} (5x^4 + 4x^3 + 2x + 3) dx$ | (D) $\int_0^{\pi} (5 + \sin x) dx$ |
| (E) $\int_{\pi}^{\pi/4} (e^{2x} + \cos x) dx$ | (F) $\int_{1.5}^{2.5} \ln x dx$ |
| (G) $\int_{0.1}^{1.0} e^x dx$ | (H) $\int_{0.2}^{1.2} \tan x dx$ |

In the numerical integration problems, the term *range of integration* denotes the entire integration range, the word *interval* denotes subdivisions of the total range of integration, and the word *increment* denotes a single increment Δx .

6.2 Direct Fit Polynomials

- Evaluate integrals (A), (B), and (C) by direct fit polynomials of order 2 and order 3 over the total range of integration. Repeat the calculations, breaking the total range of integration into two equal intervals. Compute the errors and the ratios of the errors for the two intervals.
- Evaluate integrals (D), (E), and (F) by direct fit polynomials of order 2 and order 3 over the total range of integration. Repeat the calculations, breaking the total range of integration into two equal intervals. Compute the errors and the ratios of the errors for the two intervals.
- Evaluate integrals (G) and (H) by direct fit polynomials of order 2 and order 3 over the total range of integration. Repeat the calculations, breaking the total range of integration into two equal intervals. Compute the errors and the ratios of the errors for the two intervals.

6.3 Newton-Cotes Formulas

The Trapezoid Rule

- Evaluate integrals (A), (B), and (C) by the trapezoid rule for $n = 1, 2, 4$, and 8 intervals. Compute the errors and the ratios of the errors.
- Evaluate integrals (D), (E), and (F) by the trapezoid rule for $n = 1, 2, 4$, and 8 intervals. Compute the errors and the ratios of the errors.
- Evaluate integrals (G) and (H) by the trapezoid rule for $n = 1, 2, 4$, and 8 intervals. Compute the errors and the ratios of the errors.
- Consider the function $f(x)$ tabulated in Table 1. Evaluate the integral $\int_{0.4}^{2.0} f(x) dx$ using the trapezoid rule with $n = 1, 2, 4$, and 8 intervals. The exact value is 5.86420916. Compute the errors and the ratios of the errors.

Table 1 Tabular Data

x	$f(x)$	x	$f(x)$	x	$f(x)$
0.4	5.1600	1.4	3.3886	2.2	5.7491
0.6	3.6933	1.6	3.8100	2.4	6.5933
0.8	3.1400	1.8	4.3511	2.6	7.5292
1.0	3.0000	2.0	5.0000	2.8	8.5543
1.2	3.1067				

8. Consider the function $f(x)$ tabulated in Table 1. Evaluate the integral $\int_{1.2}^{2.8} f(x) dx$ using the trapezoid rule with $n = 1, 2, 4$, and 8 intervals. The exact value is 8.43592905 . Compute the errors and the ratios of the errors.

Table 2 Tabular Data

x	$f(x)$	x	$f(x)$	x	$f(x)$
0.4	6.0900	1.4	6.9686	2.2	4.4782
0.6	7.1400	1.6	6.5025	2.4	3.6150
0.8	7.4850	1.8	5.9267	2.6	2.6631
1.0	7.5000	2.0	5.2500	2.8	1.6243
1.2	7.3100				

9. Consider the function $f(x)$ tabulated in Table 2. Evaluate the integral $\int_{0.4}^{2.0} f(x) dx$ using the trapezoid rule with $n = 1, 2, 4$, and 8 intervals. The exact value is 10.92130980 . Compute the errors and the ratios of the errors.
10. Consider the function $f(x)$ tabulated in Table 2. Evaluate the integral $\int_{1.2}^{2.8} f(x) dx$ using the trapezoid rule with $n = 1, 2, 4$, and 8 intervals.

Simpson's 1/3 Rule

11. Evaluate integrals (A), (B), and (C) using Simpson's 1/3 rule for $n = 1, 2$, and 4 intervals. Compute the errors and the ratio of the errors.
- 12*. Evaluate integrals (D), (E), and (F) using Simpson's 1/3 rule for $n = 1, 2$, and 4 intervals. Compute the errors and the ratio of the errors.
13. Evaluate integrals (G) and (H) using Simpson's 1/3 rule for $n = 1, 2$, and 4 intervals. Compute the errors and the ratio of the errors.
14. Consider the function $f(x)$ tabulated in Table 1. Evaluate the integral $\int_{0.4}^{2.0} f(x) dx$ using Simpson's 1/3 rule with $n = 1, 2$, and 4 intervals.
15. Consider the function $f(x)$ tabulated in Table 1. Evaluate the integral $\int_{1.2}^{2.8} f(x) dx$ using Simpson's 1/3 rule with $n = 1, 2$, and 4 intervals.
16. Consider the function $f(x)$ tabulated in Table 2. Evaluate the integral $\int_{0.4}^{2.0} f(x) dx$ using Simpson's 1/3 rule with $n = 1, 2$, and 4 intervals.
- 17*. Consider the function $f(x)$ tabulated in Table 2. Evaluate the integral $\int_{1.2}^{2.8} f(x) dx$ using Simpson's 1/3 rule with $n = 1, 2$, and 4 intervals.

Simpson's 3/8 Rule

18. Evaluate integrals (A), (B), and (C) using Simpson's 3/8 rule for $n = 1, 2$, and 4 intervals. Compute the errors and the ratio of the errors.
- 19*. Evaluate integrals (D), (E), and (F) using Simpson's 3/8 rule for $n = 1, 2$, and 4 intervals. Compute the errors and the ratio of the errors.
20. Evaluate integrals (G) and (H) using Simpson's 3/8 rule for $n = 1, 2$, and 4 intervals. Compute the errors and the ratio of the errors.
21. Consider the function $f(x)$ tabulated in Table 1. Evaluate the integral $\int_{0.4}^{1.6} f(x) dx$ using Simpson's 3/8 rule with $n = 1$ and 2 intervals.
22. Consider the function $f(x)$ tabulated in Table 1. Evaluate the integral $\int_{1.2}^{1.6} f(x) dx$ using Simpson's 3/8 rule with $n = 1$ and 2 intervals.
23. Consider the function $f(x)$ tabulated in Table 2. Evaluate the integral $\int_{0.4}^{1.6} f(x) dx$ using Simpson's 3/8 rule with $n = 1$ and 2 intervals.

24. Consider the function $f(x)$ tabulated in Table 2. Evaluate the integral $\int_{1.2}^{1.6} f(x) dx$ using Simpson's 3/8 rule with $n = 1$ and 2 intervals.
25. Evaluate integrals (A), (B), and (C) using Simpson's rules with $n = 5$ and 7 increments.
26. Evaluate integrals (D), (E), and (F) using Simpson's rules with $n = 5$ and 7 increments.
27. Evaluate integrals (G) and (H) using Simpson's rules with $n = 5$ and 7 increments.

Higher-Order Newton-Cotes Formulas

28. Derive the Newton-Cotes formulas for polynomials of degree $n = 4, 5, 6$, and 7.
29. Evaluate integrals (D), (E), and (F) by the Newton-Cotes fourth-order formula with one and two intervals. Compute the errors and the ratios of the errors.
30. Evaluate integrals (D), (E), and (F) by the Newton-Cotes fifth-order formula with one and two intervals. Compute the errors and the ratios of the errors.
31. Evaluate integrals (D), (E), and (F) by the Newton-Cotes sixth-order formula with one and two intervals. Compute the errors and the ratios of the errors.
32. Evaluate integrals (D), (E), and (F) by the Newton-Cotes seventh-order formula with one and two intervals. Compute the errors and the ratios of the errors.

6.4 Extrapolation and Romberg Integration

33. Evaluate the following integrals using Romberg integration with four intervals. Let the first interval be the total range of integration.
 - (a) $\int_0^{\pi/4} \tan x dx$
 - (b) $\int_0^{0.5} e^{-x} dx$
 - (c) $\int_0^{3/4} e^{-x^2} dx$
 - (d) $\int_1^{2.5} (x^5 - x^2) dx$
34. Evaluate integrals (A), (B), and (C) using Romberg integration with four intervals. Let the first interval be the total range of integration.
- 35*. Evaluate integrals (D), (E), and (F) using Romberg integration with four intervals. Let the first interval be the total range of integration.
36. Evaluate integrals (G) and (H) using Romberg integration with four intervals. Let the first interval be the total range of integration.
37. Consider the function $f(x)$ tabulated in Table 1. Evaluate the integral $\int_{0.4}^{2.0} f(x) dx$ using Romberg integration with $n = 1, 2, 4$, and 8 intervals.
38. Consider the function $f(x)$ tabulated in Table 2. Evaluate the integral $\int_{0.4}^{2.0} f(x) dx$ using Romberg integration with $n = 1, 2, 4$, and 8 intervals.
39. Which row of a Romberg table yields $\int_a^b f(x) dx$ exactly if $f(x)$ is a polynomial of degree k , where (a) $k = 3$, (b) $k = 5$, and (c) $k = 7$? Verify your conclusion for $\int_0^2 x^k dx$. Start with one interval.

6.5 Adaptive Integration

40. Evaluate $I = \int_{-2.0}^{0.4} e^{-x} dx$ using Romberg integration. (a) Let the first interval be the total range of integration. (b) Divide the total range of integration into two equal intervals and use Romberg integration in each interval. Let the first subinterval in each interval be the total range of integration for that interval. (c) Divide the total range of integration into the two intervals, $-2.0 \leq$

$x \leq -0.4$ and $-0.4 \leq x \leq 0.4$, and repeat part (b). (d) Compare the errors incurred for the three cases.

41. Evaluate $I = \int_1^3 \ln(x) dx$ using Romberg integration. (a) Start with the total interval. Let the first subinterval be the total interval. (b) Divide the total range of integration into two equal intervals and use Romberg integration in each interval. Let the first subinterval in each interval be the total interval.

6.6 Gaussian Quadrature

42. Evaluate integrals (A), (B), and (C) by two-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
- 43*. Evaluate integrals (D), (E), and (F) by two-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
44. Evaluate integrals (G) and (H) by two-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
45. Evaluate integrals (A), (B), and (C) by three-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
- 46*. Evaluate integrals (D), (E), and (F) by three-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
47. Evaluate integrals (G) and (H) by three-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
48. Evaluate integrals (A), (B), and (C) by four-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
- 49*. Evaluate integrals (D), (E), and (F) by four-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
50. Evaluate integrals (G) and (H) by four-point Gaussian quadrature for $n = 1, 2$, and 4 intervals. Compute the errors and the ratios of the errors.
51. Evaluate the following integrals using k -point Gaussian quadrature for $k = 2, 3$, and 4, with $n = 1, 2$, and 4 intervals. Compare the results with the exact solutions.
- (a) $\int_1^{2.6} xe^{-x^2} dx$ (b) $\int_1^3 \cosh x dx$ (c) $\int_0^4 \sinh x dx$ (d) $\int_{-1}^2 e^{-x} \sin x dx$
52. (a) What is the smallest k for which k -point Gaussian quadrature is exact for a polynomial of degree 7? (b) Verify the answer to part (a) by evaluating $\int_0^2 x^7 dx$.

6.7 Multiple Integrals

53. Evaluate the multiple integral $\int_{-1}^1 \int_0^2 (4x^3 - 2x^2y + 3xy^2) dx dy$: (a) analytically, (b) using the trapezoid rule, and (c) using Simpson's 1/3 rule.
54. Evaluate the multiple integral $\int_0^1 \int_0^2 \sin(x^2 + y^2) dx dy$: (a) analytically, (b) using the trapezoid rule, and (c) using Simpson's 1/3 rule.
55. Evaluate the multiple integral $\int_0^1 \int_0^{e^x} (x^2 + 1/y) dy dx$: (a) analytically, (b) using the trapezoid rule for both integrals, (c) using three-point Gaussian quadrature for both integrals; (d) using the trapezoid rule for the y integral and three-point Gaussian quadrature for the x integral, and (e) using three-point Gaussian quadrature for the y integral and the trapezoid rule for the x integral.
56. Evaluate the multiple integral $\int_{-1}^1 \int_0^{x^2} xy dy dx$ by the procedures described in the previous problem.

6.8 Programs

57. Implement the trapezoid rule program presented in Section 6.8.1. Check out the program using the given data set.
58. Solve any of Problems 4 to 10 using the program.
59. Implement the Simpson's 1/3 rule program presented in Section 6.8.2. Check out the program using the given data set.
60. Solve any of Problems 11 to 17 using the program.
61. Implement the Simpson's 3/8 rule program presented in Section 6.8.3. Check out the program using the given data set.
62. Solve any of Problems 18 to 24 using the program.

APPLIED PROBLEMS

63. Evaluate the integral $\iint_R \exp[(x + y^2)^{1/2}] dy dx$, where R is the area enclosed by the circle $x^2 + y^2 = 1$. Use Cartesian coordinates.
64. Find the volume of a circular pyramid with height and base radius equal to 1. Use Cartesian coordinates.