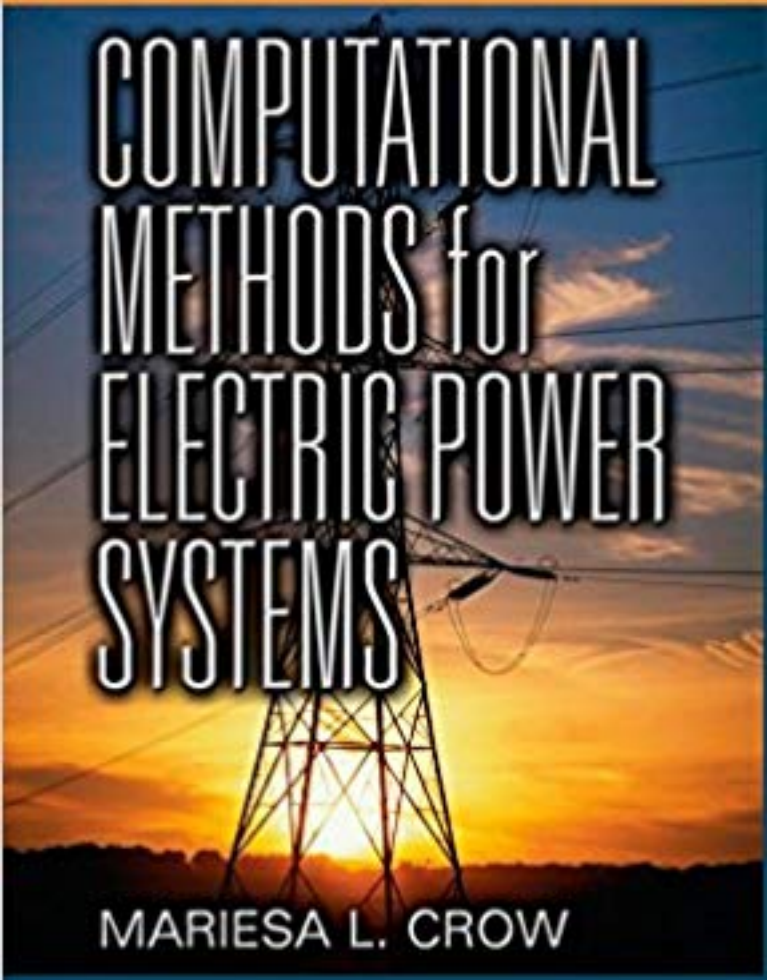
SECOND EDITION

COMPUTATIONAL METHODS for ELECTRIC POWER SYSTEMS



MARIESA L. CROW

CRC Press
Taylor & Francis Group

COMPUTATIONAL METHODS for ELECTRIC POWER SYSTEMS

SECOND EDITION

The ELECTRIC POWER ENGINEERING Series
Series Editor Leo L. Grigsby

Published Titles

Computational Methods for Electric Power Systems, Second Edition
Mariesa L. Crow

Electric Energy Systems: Analysis and Operation
Antonio Gómez-Expósito, Antonio J. Conejo, and Claudio Cañizares

Distribution System Modeling and Analysis, Second Edition
William H. Kersting

Electric Machines
Charles A. Gross

Harmonics and Power Systems
Francisco C. De La Rosa

Electric Drives, Second Edition
Ion Boldea and Syed Nasar

Power System Operations and Electricity Markets
Fred I. Denny and David E. Dismukes

Power Quality
C. Sankaran

Electromechanical Systems, Electric Machines,and Applied Mechatronics
Sergey E. Lyshevski

Linear Synchronous Motors: Transportation and Automation Systems
Jacek Gieras and Jerry Piech

Electrical Energy Systems, Second Edition
Mohamed E. El-Hawary

The Induction Machine Handbook
Ion Boldea and Syed Nasar

Electric Power Substations Engineering
John D. McDonald

Electric Power Transformer Engineering
James H. Harlow

Electric Power Distribution Handbook
Tom Short

SECOND EDITION

COMPUTATIONAL METHODS for ELECTRIC POWER SYSTEMS

MARIESA L. CROW

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-13: 978-1-4200-8661-4 (Ebook-PDF)

To
Jim, David, and Jacob

Preface to the Second Edition

This new edition has been updated to include new material. Specifically, this new edition has added sections on the following material:

- Generalized Minimal Residual (GMRES) methods
- Numerical di erentiation
- Secant method
- Homotopy and continuation methods
- Power method for computing dominant eigenvalues
- Singular-value decomposition and pseudoinverses
- Matrix pencil method

and a significant revision of the Optimization chapter (Chapter 6) to include linear and quadratic programming methods.

A course structure would typically include the following chapters in sequence: Chapter 1, 2, and 3. From this point, any of the chapters can follow without loss of consistency. I have tried to structure each chapter to give the reader an overview of the methods with salient examples. In many cases however, it is not possible to give an exhaustive coverage of the material; many topics have decades of work devoted to their development.

Many of the methods presented in this book have commercial software packages that will accomplish their solution far more rigorously with many failsafe attributes included (such as accounting for ill-conditioning, etc.). It is not my intent to make students experts in each topic, but rather to develop an appreciation for the methods behind the packages. Many commercial packages provide default settings or choices of parameters for the user; through better understanding of the methods driving the solution, informed users can make better choices and have a better understanding of the situations in which the methods may fail. If this book provides any reader with more confidence in using commercial packages, I have succeeded in my intent.

As before, I am indebted to many people: my husband Jim and my children David and Jacob for making every day a joy, my parents Lowell and Sondra for their continuing support, and Frieda Adams for all she does to help me succeed.

Mariesa L. Crow
Rolla, Missouri
2009

Preface to the First Edition

This book is the outgrowth of a graduate course that I've taught at the University of Missouri-Rolla for the past decade or so. Over the years, I've used a number of excellent textbooks for this course, but each textbook was always missing some of the topics that I wanted to cover in the class. After relying on handouts for many years, my good friend Leo Grigsby encouraged me to put them down in the form of a book (if arm-twisting can be called encouragement . . .). With the support of my graduate students, who I used as testbeds for each chapter, this book gradually came into existence. I hope that those who read this book will find this field as stimulating as I have found it.

In addition to Leo and the fine people at CRC Press, I'm grateful to the University of Missouri-Rolla administration and the Department of Electrical and Computer Engineering for providing the environment to nurture my teaching and research and giving me the latitude to pursue my personal interests in this field.

Lastly, I don't often get the opportunity to publicly acknowledge the people who've been instrumental in my professional development. I'd like to thank: Marija Ilic, who initially put me on the path; Peter Sauer, who encouraged me along the way; Jerry Heydt, for providing inspiration; Frieda Adams, for all she does to make my life easier; Steve Pekarek, for putting up with my grumbling and complaining; and Lowell and Sondra Crow for making it all possible.

Mariesa L. Crow
Rolla, Missouri
2003

Contents

1

Introduction

In today's deregulated environment, the nation's electric power network is being forced to operate in a manner for which it was not intentionally designed. Therefore, system analysis is very important to predict and continually update the operating status of the network. This includes estimating the current power flows and bus voltages (Power Flow Analysis and State Estimation), determining the stability limits of the system (Continuation Power Flow, Numerical Integration for Transient Stability, and Eigenvalue Analysis), and minimizing costs (Optimal Power Flow). This book provides an introductory study of the various computational methods that form the basis of many analytical studies in power systems and other engineering and science fields. This book provides the analytical background of the algorithms used in numerous commercial packages. By understanding the theory behind many of the algorithms, the reader/user can better use the software and make more informed decisions (i.e., choice of integration method and step-size in simulation packages).

Due to the sheer size of the power grid, hand-based calculations are nearly impossible and computers o er the only truly viable means for system analysis. The power industry is one of the largest users of computer technology and one of the first industries to embrace the potential of computer analysis when mainframes first became available. Although the first algorithms for power system analysis were developed in the 1940's, it wasn't until the 1960's when computer usage became widespread within the power industry. Many of the analytical techniques and algorithms used today for the simulation and analysis of large systems were originally developed for power system applications.

As power systems increasingly operate under stressed conditions, computer simulation will play a large role in control and security assessment. Commercial packages routinely fail or give erroneous results when used to simulate stressed systems. Understanding of the underlying numerical algorithms is imperative to correctly interpret the results of commercial packages. For example, will the system really exhibit the simulated behavior or is the simulation simply an artifact of a numerical inaccuracy? The educated user can make better judgments about how to compensate for numerical shortcomings in such packages, either by better choice of simulation parameters or by posing the problem in a more numerically tractable manner. This book will provide the background for a number of widely used numerical algorithms that

underlie many commercial packages for power system analysis and design.

This book is intended to be used as a text in conjunction with a semester-long graduate level course in computational algorithms. While the majority of examples in this text are based on power system applications, the theory is presented in a general manner so as to be applicable to a wide range of engineering systems. Although some knowledge of power system engineering may be required to fully appreciate the subtleties of some of the illustrations, such knowledge is not a prerequisite for understanding the algorithms themselves. The text and examples are used to provide an introduction to a wide range of numerical methods without being an exhaustive reference. Many of the algorithms presented in this book have been the subject of numerous modifications and are still the object of on-going research. As this text is intended to provide a foundation, many of these new advances are not explicitly covered, but are rather given as references for the interested reader. The examples in this text are intended to be simple and thorough enough to be reproduced easily. Most "real world" problems are much larger in size and scope, but the methodologies presented in this text should su ciently prepare the reader to cope with any di culties he/she may encounter.

Most of the examples in this text were produced using code written in Matlab^{TM}. Although this was the platform used by the author, in practice, any computer language may be used for implementation. There is no practical reason for a preference for any particular platform or language.

2

The Solution of Linear Systems

In many branches of engineering and science it is desirable to be able to mathematically determine the state of a system based on a set of physical relationships. These physical relationships may be determined from characteristics such as circuit topology, mass, weight, or force to name a few. For example, the injected currents, network topology, and branch impedances govern the voltages at each node of a circuit. In many cases, the relationship between the known, or input, quantities and the unknown, or output, states is a linear relationship. Therefore, a linear system may be generically modeled as

$$Ax = b \tag{2.1}$$

where b is the $n \times 1$ vector of known quantities, x is the $n \times 1$ unknown state vector, and A is the $n \times n$ matrix that relates x to b. For the time being, it will be assumed that the matrix A is invertible, or non-singular; thus, each vector b will yield a unique corresponding vector x. Thus the matrix A^{-1} exists and

$$x^* = A^{-1}b \tag{2.2}$$

is the unique solution to equation (2.1).

The natural approach to solving equation (2.1) is to directly calculate the inverse of A and multiply it by the vector b. One method to calculate A^{-1} is to use *Cramer's* rule :

$$A^{-1}(i,j) = \frac{1}{det(A)} (A_{ij})^T \quad \text{for } i = 1,\ldots,n, j = 1,\ldots,n \tag{2.3}$$

where $A^{-1}(i,j)$ is the ij^{th} entry of A^{-1} and A_{ij} is the cofactor of each entry a_{ij} of A. This method requires the calculation of $(n + 1)$ determinants which results in $2(n + 1)!$ multiplications to find A^{-1}! For large values of n, the calculation requirement grows too rapidly for computational tractability; thus, alternative approaches have been developed.

Basically there are two approaches to solving equation (2.1):

- The *direct methods*, or elimination methods, find the exact solution (within the accuracy of the computer) through a finite number of arithmetic operations. The solution x of a direct method would be completely accurate were it not for computer roundo errors.

- *Iterative methods*, on the other hand, generate a sequence of (hopefully) progressively improving approximations to the solution based on the application of the same computational procedure at each step. The iteration is terminated when an approximate solution is obtained having some pre-specified accuracy or when it is determined that the iterates are not improving.

The choice of solution methodology usually relies on the structure of the system under consideration. Certain systems lend themselves more amenably to one type of solution method versus the other. In general, direct methods are best for full matrices, whereas iterative methods are better for matrices that are large and sparse. But as with most generalizations, there are notable exceptions to this rule of thumb.

2.1 Gaussian Elimination

An alternate method for solving equation (2.1) is to solve for x without calculating A^{-1} explicitly. This approach is a *direct method* of linear system solution, since x is found directly. One common direct method is the method of *Gaussian elimination*. The basic idea behind Gaussian elimination is to use the first equation to eliminate the first unknown from the remaining equations. This process is repeated sequentially for the second unknown, the third unknown, etc., until the elimination process is completed. The n-th unknown is then calculated directly from the input vector b. The unknowns are then recursively substituted back into the equations until all unknowns have been calculated.

Gaussian elimination is the process by which the augmented $n \times (n + 1)$ matrix

$$[A \mid b]$$

is converted to the $n \times (n + 1)$ matrix

$$[I \mid b^*]$$

through a series of elementary row operations, where

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ Ix &= A^{-1}b = b^* \\ x^* &= b^* \end{aligned}$$

Thus if a series of elementary row operations exist that can transform the matrix A into the identity matrix I, then the application of the same set of

elementary row operations will also transform the vector b into the solution vector x^*.

An elementary row operation consists of one of three possible actions that can be applied to a matrix:

- interchange any two rows of the matrix
- multiply any row by a constant
- take a linear combination of rows and add it to another row

The elementary row operations are chosen to transform the matrix A into an upper triangular matrix that has ones on the diagonal and zeros in the sub-diagonal positions. This process is known as the *forward elimination* step. Each step in the forward elimination can be obtained by successively pre-multiplying the matrix A by an elementary matrix ξ, where ξ is the matrix obtained by performing an elementary row operation on the identity matrix.

Example 2.1

Find a sequence of elementary matrices that when applied to the following matrix will produce an upper triangular matrix.

$$A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix}$$

Solution 2.1 To upper triangularize the matrix, the elementary row operations will need to systematically zero out each column below the diagonal. This can be achieved by replacing each row of the matrix below the diagonal with the di erence of the row itself and a constant times the diagonal row, where the constant is chosen to result in a zero sum in the column under the diagonal. Therefore row 2 of A is replaced by (row 2 - 2(row 1)) and the elementary matrix is

$$\xi_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$\xi_1 A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 0 & -5 & -6 & -13 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix}$$

Note that all rows except row 2 remain the same and row 2 now has a 0 in the column under the first diagonal. Similarly the two elementary matrices that complete the elimination of the first column are:

$$\xi_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -4 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\xi_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -9 & 0 & 0 & 1 \end{bmatrix}$$

and

$$\xi_3\xi_2\xi_1 A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 0 & -5 & -6 & -13 \\ 0 & -9 & -11 & -24 \\ 0 & -25 & -29 & -68 \end{bmatrix} \tag{2.4}$$

The process is now applied to the second column to zero out everything below the second diagonal and scale the diagonal to one. Therefore

$$\xi_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -\frac{9}{5} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\xi_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{25}{5} & 0 & 1 \end{bmatrix}$$

$$\xi_6 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly,

$$\xi_6\xi_5\xi_4\xi_3\xi_2\xi_1 A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 0 & 1 & \frac{6}{5} & \frac{13}{5} \\ 0 & 0 & -\frac{1}{5} & -\frac{3}{5} \\ 0 & 0 & 1 & -3 \end{bmatrix} \tag{2.5}$$

Similarly,

$$\xi_7 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5 & 1 \end{bmatrix}$$

$$\xi_8 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

yielding

$$\xi_8\xi_7\xi_6\xi_5\xi_4\xi_3\xi_2\xi_1 A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 0 & 1 & \frac{6}{5} & \frac{13}{5} \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & -6 \end{bmatrix} \tag{2.6}$$

Lastly,

$$\xi_9 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -\frac{1}{6} \end{bmatrix}$$

and

$$\xi_9\xi_8\xi_7\xi_6\xi_5\xi_4\xi_3\xi_2\xi_1 A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 0 & 1 & \frac{6}{5} & \frac{13}{5} \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

which completes the upper triangularization process.

Once an upper triangular matrix has been achieved, the solution vector x^* can be found by successive substitution (or *back substitution*) of the states.

Example 2.2

Using the upper triangular matrix of Example 2.1, find the solution to

$$\begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Solution 2.2 Note that the product of a series of lower triangular matrices is lower triangular; therefore, the product

$$W = \xi_9\xi_8\xi_7\xi_6\xi_5\xi_4\xi_3\xi_2\xi_1 \tag{2.8}$$

is lower triangular. Since the application of the elementary matrices to the matrix A results in an upper triangular matrix, then

$$WA = U \tag{2.9}$$

where U is the upper triangular matrix that results from the forward elimination process. Premultiplying equation (2.1) by W yields

$$WAx = Wb \tag{2.10}$$

$$Ux = Wb \tag{2.11}$$

$$= b' \tag{2.12}$$

where $Wb = b'$.

From Example 2.1:

$$W = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{2}{5} & -\frac{1}{5} & 0 & 0 \\ 2 & 9 & -5 & 0 \\ \frac{1}{6} & \frac{14}{6} & -\frac{5}{6} & -\frac{1}{6} \end{bmatrix}$$

and

$$b' = W \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{5} \\ 6 \\ \frac{3}{2} \end{bmatrix}$$

Thus,

$$\begin{bmatrix} 1 & 3 & 4 & 8 \\ 0 & 1 & \frac{6}{5} & \frac{13}{5} \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{5} \\ 6 \\ \frac{3}{2} \end{bmatrix} \tag{2.13}$$

By inspection, $x_4 = \frac{3}{2}$. The third row yields

$$x_3 = 6 - 3x_4 \tag{2.14}$$

Substituting the value of x_4 into equation (2.14) yields $x_3 = \frac{3}{2}$. Similarly,

$$x_2 = \frac{1}{5} - \frac{6}{5}x_3 - \frac{13}{5}x_4 \tag{2.15}$$

and substituting x_3 and x_4 into equation (2.15) yields $x_2 = -\frac{11}{2}$. Solving for x_1 in a similar manner produces

$$x_1 = 1 - 3x_2 - 4x_3 - 8x_4 \tag{2.16}$$

$$= -\frac{1}{2} \tag{2.17}$$

Thus,

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -1 \\ -11 \\ 3 \\ 3 \end{bmatrix}$$

The solution methodology of successively substituting values of x back into the equation as they are found gives rise to the name of *back substitution* for this step of the Gaussian elimination. Therefore, Gaussian elimination consists of two main steps: forward elimination and back substitution. Forward elimination is the process of transforming the matrix A into triangular factors. Back substitution is the process by which the unknown vector x is found from the input vector b and the factors of A. Gaussian elimination also provides the framework under which the LU factorization process is developed.

2.2 LU Factorization

The forward elimination step of Gaussian elimination produces a series of upper and lower triangular matrices that are related to the A matrix as given in equation (2.9). The matrix W is a lower triangular matrix and U is an upper triangular matrix with ones on the diagonal. Recall that the inverse of a lower triangular matrix is also a lower triangular matrix; therefore, if

$$L \triangleq W^{-1}$$

then

$$A = LU$$

The matrices L and U give rise to the name of the factorization/elimination algorithm known as "LU factorization." In fact, given any nonsingular matrix A, there exists some permutation matrix P (possibly $P = I$), such that

$$LU = PA \tag{2.18}$$

where U is upper triangular with unit diagonals, L is lower triangular with nonzero diagonals, and P is a matrix of ones and zeros obtained by rearranging the rows and columns of the identity matrix. Once a proper matrix P is chosen, this factorization is unique [6]. Once $P, L,$ and U are determined, then the system

$$Ax = b \tag{2.19}$$

can be solved expeditiously. Premultiplying equation (2.19) by the matrix P yields

$$PAx = Pb = b' \tag{2.20}$$

$$LUx = b' \tag{2.21}$$

where b' is just a rearrangement of the vector b. Introducing a "dummy" vector y such that

$$Ux = y \tag{2.22}$$

thus

$$Ly = b' \tag{2.23}$$

Consider the structure of equation (2.23):

$$\begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{bmatrix}$$

The elements of the vector y can be found by straightforward substitution:

$$\begin{aligned} y_1 &= \frac{b'_1}{l_{11}} \\ y_2 &= \frac{1}{l_{22}}(b'_2 - l_{21}y_1) \\ y_3 &= \frac{1}{l_{33}}(b'_3 - l_{31}y_1 - l_{32}y_2) \\ &\vdots \\ y_n &= \frac{1}{l_{nn}}\left(b'_n - \sum_{j=1}^{n-1} l_{nj}y_j\right) \end{aligned}$$

After the vector y has been found, then x can be easily found from

$$\begin{bmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & 1 & u_{23} & \cdots & u_{2n} \\ 0 & 0 & 1 & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \vdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

Similarly, the solution vector x can be found by backward substitution:

$$\begin{aligned} x_n &= y_n \\ x_{n-1} &= y_{n-1} - u_{n-1,n}x_n \\ x_{n-2} &= y_{n-2} - u_{n-2,n}x_n - u_{n-2,n-1}x_{n-1} \\ &\vdots \\ x_1 &= y_1 - \sum_{j=2}^{n} u_{1j}x_j \end{aligned}$$

The value of LU factorization is that once A is factored into the upper and lower triangular matrices, the solution for the solution vector x is straightforward. Note that the inverse to A is never explicitly found.

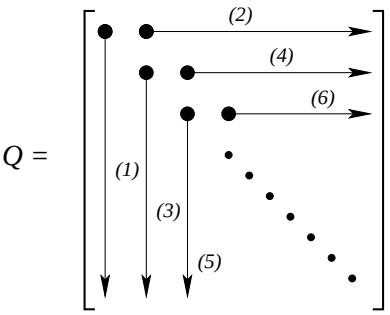


FIGURE 2.1
Order of calculating columns and rows of Q

Several methods for computing the LU factors exist and each method has its advantages and disadvantages. One common factorization approach is known as the *Crout's* algorithm for finding the LU factors [6]. Let the matrix Q be defined as

$$Q \triangleq L + U - I = \begin{bmatrix} l_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21} & l_{22} & u_{23} & \cdots & u_{2n} \\ l_{31} & l_{32} & l_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \tag{2.24}$$

Crout's algorithm computes the elements of Q first by column and then row as shown in Figure 2.1. Each element q_{ij} of Q depends only on the a_{ij} entry of A and previously computed values of Q.

Crout's Algorithm for Computing LU from A

1. Initialize Q to the zero matrix. Let $j = 1$.
2. Complete the j^{th} column of Q (j^{th} column of L) as

$$q_{kj} = a_{kj} - \sum_{i=1}^{j-1} q_{ki} q_{ij} \quad \text{for } k = j, \ldots, n \tag{2.25}$$

3. If $j = n$, then stop.
4. Assuming that $q_{jj} = 0$, complete the j^{th} row of Q (j^{th} row of U) as

$$q_{jk} = \frac{1}{q_{jj}} \left(a_{jk} - \sum_{i=1}^{j-1} q_{ji} q_{ik} \right) \quad \text{for } k = j + 1, \ldots, n \tag{2.26}$$

5. Set $j = j + 1$. Go to step 2.

Once the LU factors are found, then the dummy vector y can be found by forward substitution:

$$y_k = \frac{1}{q_{kk}} \left(b_k - \sum_{j=1}^{k-1} q_{kj} y_j \right) \quad \text{for } k = 1, \ldots, n \tag{2.27}$$

Similarly, the solution vector x can be found by backward substitution:

$$x_k = y_k - \sum_{j=k+1}^{n} q_{kj} x_j \quad \text{for } k = n, n-1, \ldots, 1 \tag{2.28}$$

One measure of the computation involved in the LU factorization process is to count the number of multiplications and divisions required to find the solution since these are both floating point operations. Computing the j^{th} column of Q (j^{th} column of L) requires

$$\sum_{j=1}^{n} \sum_{k=j}^{n} (j-1)$$

multiplications and divisions. Similarly, computing the j^{th} row of Q (j^{th} row of U) requires

$$\sum_{j=1}^{n-1} \sum_{k=j+1}^{n} j$$

multiplications and divisions. The forward substitution step requires

$$\sum_{j=1}^{n} j$$

and the backward substitution step requires

$$\sum_{j=1}^{n} (n-j)$$

multiplications and divisions. Taken together, the LU factorization procedure requires

$$\frac{1}{3}\left(n^3 - n\right)$$

and the substitution steps require n^2 multiplications and divisions. Therefore the whole process of solving the linear system of equation (2.1) requires a total of

$$\frac{1}{3}\left(n^3 - n\right) + n^2 \tag{2.29}$$

multiplications and divisions. Compare this to the requirements of Cramer's rule which requires 2(n+1)! multiplications and divisions. Obviously for a system of any significant size, it is far more computationally e cient to use LU factorization and forward/backward substitution to find the solution x.

Example 2.3

Using LU factorization with forward and backward substitution, find the solution to the system of Example 2.2.

Solution 2.3 The first step is to find the LU factors of the A matrix:

$$A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix}$$

Starting with $j = 1$, equation (2.25) indicates that the elements of the first column of Q are identical to the elements of the first column of A. Similarly, according to equation (2.26), the first row of Q becomes:

$$q_{12} = \frac{a_{12}}{q_{11}} = \frac{3}{1} = 3$$
$$q_{13} = \frac{a_{13}}{q_{11}} = \frac{4}{1} = 4$$
$$q_{14} = \frac{a_{14}}{q_{11}} = \frac{8}{1} = 8$$

Thus for $j = 1$, the Q matrix becomes:

$$Q = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & & & \\ 4 & & & \\ 9 & & & \end{bmatrix}$$

For $j = 2$, the second column and row of Q below and to the right of the diagonal, respectively, will be calculated. For the second column of Q:

$$q_{22} = a_{22} - q_{21}q_{12} = 1 - (2)(3) = -5$$
$$q_{32} = a_{32} - q_{31}q_{12} = 3 - (4)(3) = -9$$
$$q_{42} = a_{42} - q_{41}q_{12} = 2 - (9)(3) = -25$$

Each element of Q uses the corresponding element of A and elements of Q that have been previously computed. Note also that the inner indices of the products are always the same and the outer indices are the same as the indices of the element being computed. This holds true for both column and

row calculations. The second row of Q is computed:

$$q_{23} = \frac{1}{q_{22}}(a_{23} - q_{21}q_{13}) = \frac{1}{-5}(2 - (2)(4)) = \frac{6}{5}$$
$$q_{24} = \frac{1}{q_{22}}(a_{24} - q_{21}q_{14}) = \frac{1}{-5}(3 - (2)(8)) = \frac{13}{5}$$

After $j = 2$, the Q matrix becomes:

$$Q = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & -5 & \frac{6}{5} & \frac{13}{5} \\ 4 & -9 & & \\ 9 & -25 & & \end{bmatrix}$$

Continuing on for $j = 3$, the third column of Q is calculated

$$q_{33} = a_{33} - (q_{31}q_{13} + q_{32}q_{23}) = 5 - \left((4)(4) + (-9)\frac{6}{5}\right) = -\frac{1}{5}$$
$$q_{43} = a_{43} - (q_{41}q_{13} + q_{42}q_{23}) = 7 - \left((9)(4) + (-25)\frac{6}{5}\right) = 1$$

and the third row of Q becomes

$$q_{34} = \frac{1}{q_{33}}(a_{34} - (q_{31}q_{14} + q_{32}q_{24}))$$
$$= (-5)\left(8 - \left((4)(8) + (-9)\left(\frac{13}{5}\right)\right)\right) = 3$$

yielding

$$Q = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & -5 & \frac{6}{5} & \frac{13}{5} \\ 4 & -9 & -\frac{1}{5} & 3 \\ 9 & -25 & 1 & \end{bmatrix}$$

Lastly, for $j = 4$, the final diagonal element is found:

$$q_{44} = a_{44} - (q_{41}q_{14} + q_{42}q_{24} + q_{43}q_{34})$$
$$= 4 - \left((9)(8) + (-25)\left(\frac{13}{5}\right) + (3)(1)\right) = -6$$

Thus:

$$Q = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & -5 & \frac{6}{5} & \frac{13}{5} \\ 4 & -9 & -\frac{1}{5} & 3 \\ 9 & -25 & 1 & -6 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & -5 & 0 & 0 \\ 4 & -9 & -\frac{1}{5} & 0 \\ 9 & -25 & 1 & -6 \end{bmatrix}$$

$$U = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 0 & 1 & \frac{6}{5} & \frac{13}{5} \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

One method of checking the correctness of the solution is to check if $LU = A$, which in this case it does.

Once the LU factors have been found, then the next step in the solution process is the forward elimination using the L matrix and the b vector to find the dummy vector y. Using forward substitution to solve $Ly = b$ for y:

$$y_1 = \frac{b_1}{L_{11}} = \frac{1}{1} = 1$$

$$y_2 = \frac{(b_2 - L_{21}y_1)}{L_{22}} = \frac{(1 - (2)(1))}{-5} = \frac{1}{5}$$

$$y_3 = \frac{(b_3 - (L_{31}y_1 + L_{32}y_2))}{L_{33}} = (-5)\left(1 - \left((4)(1) + (-9)\frac{1}{5}\right)\right) = 6$$

$$y_4 = \frac{(b_4 - (L_{41}y_1 + L_{42}y_2 + L_{43}y_3))}{L_{44}}$$

$$= \frac{\left(1 - \left((9)(1) + (-25)\left(\frac{1}{5}\right) + (1)(6)\right)\right)}{-6} = \frac{3}{2}$$

Thus

$$y = \begin{bmatrix} 1 \\ 1 \\ 5 \\ 6 \\ 3 \\ 2 \end{bmatrix}$$

Similarly, backward substitution is then applied to $Ux = y$ to find the solution vector x:

$$x_4 = y_4 = \frac{3}{2}$$

$$x_3 = y_3 - U_{34}x_4 = 6 - (3)\left(\frac{3}{2}\right) = \frac{3}{2}$$

$$x_2 = y_2 - (U_{24}x_4 + U_{23}x_3) = \frac{1}{5} - \left(\left(\frac{13}{5}\right)\left(\frac{3}{2}\right) + \left(\frac{6}{5}\right)\left(\frac{3}{2}\right)\right) = -\frac{11}{2}$$

$$x_1 = y_1 - (U_{14}x_4 + U_{13}x_3 + U_{12}x_2)$$

$$= 1 - \left((8)\left(\frac{3}{2}\right) + (4)\left(\frac{3}{2}\right) + (3)\left(-\frac{11}{2}\right)\right) = -\frac{1}{2}$$

yielding the final solution vector

$$x = \frac{1}{2}\begin{bmatrix} -1 \\ -11 \\ 3 \\ 3 \end{bmatrix}$$

which is the same solution found by Gaussian elimination and backward substitution in Example 2.2. A quick check to verify the correctness of the solution is to substitute the solution vector x back into the linear system $Ax = b$.

2.2.1 LU Factorization with Partial Pivoting

The LU factorization process presented assumes that the diagonal element is non-zero. Not only must the diagonal element be non-zero, it must be on the same order of magnitude as the other non-zero elements. Consider the solution of the following linear system:

$$\begin{bmatrix} 10^{-10} & 1 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \end{bmatrix} \tag{2.30}$$

By inspection, the solution to this linear system is

$$x_1 \approx 2$$
$$x_2 \approx 1$$

The LU factors for A are

$$L = \begin{bmatrix} 10^{-10} & 0 \\ 2 & \left(1 - 2 \times 10^{10}\right) \end{bmatrix}$$
$$U = \begin{bmatrix} 1 & 10^{10} \\ 0 & 1 \end{bmatrix}$$

Applying forward elimination to solve for the dummy vector y yields:

$$y_1 = 10^{10}$$
$$y_2 = \frac{\left(5 - 2 \times 10^{10}\right)}{\left(1 - 2 \times 10^{10}\right)} \approx 1$$

Back substituting y into $Ux = y$ yields

$$x_2 = y_2 \approx 1$$
$$x_1 = 10^{10} - 10^{10}x_2 \approx 0$$

The solution for x_2 is correct, but the solution for x_1 is considerably o . Why did this happen? The problem with the equations arranged the way they are

in equation (2.30) is that 10^{-10} is too near zero for most computers. However, if the equations are rearranged such that

$$\begin{bmatrix} 2 & 1 \\ 10^{-10} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix} \tag{2.31}$$

then the LU factors become

$$L = \begin{bmatrix} 2 & 0 \\ 10^{-10} & \left(1 - \frac{1}{2} \times 10^{-10}\right) \end{bmatrix}$$
$$U = \begin{bmatrix} 1 & \frac{1}{2} \\ 0 & 1 \end{bmatrix}$$

The dummy vector y becomes

$$y_1 = 5$$
$$y_2 = \frac{\left(1 - \frac{5}{2} \times 10^{-10}\right)}{\left(1 - \frac{1}{2} \times 10^{-10}\right)} \approx 1$$

and by back substitution, x becomes

$$x_2 \approx 1$$
$$x_1 \approx \frac{5}{2} - \frac{1}{2}(1) = 2$$

which is the solution obtained by inspection of the equations. Therefore even though the diagonal entry may not be exactly zero, it is still good practice to rearrange the equations such that the largest magnitude element lies on the diagonal. This process is known as *pivoting* and gives rise to the permutation matrix P of equation (2.18).

Since the Crout's algorithm computes the Q matrix by column and row with increasing index, only *partial pivoting* can used, that is, only the rows of Q (and correspondingly A) can be exchanged. The columns must remain static. To choose the best pivot, the column beneath the j^{th} diagonal (at the j^{th} step in the LU factorization) is searched for the element with the largest absolute value. The corresponding row and the j^{th} row are then exchanged. The pivoting strategy may be succinctly expressed as:

Partial Pivoting Strategy

1. At the j^{th} step of LU factorization, choose the k^{th} row as the exchange row such that
$$|q_{jj}| = max\,|q_{kj}| \text{ for } k = j, \ldots, n \tag{2.32}$$

2. Exchange rows and update A, P, and Q correspondingly.

The permutation matrix P is comprised of ones and zeros and is obtained as the product of a series of elementary permutation matrices $P^{j,k}$ which represent the exchange of rows j and k. The elementary permutation matrix $P^{j,k}$, shown in Figure 2.2, is obtained from the identify matrix by interchanging rows j and k. A pivot is achieved by the pre-multiplication of a properly chosen $P^{j,k}$. Since this is only an interchange of rows, the order of the unknown vector does not change.

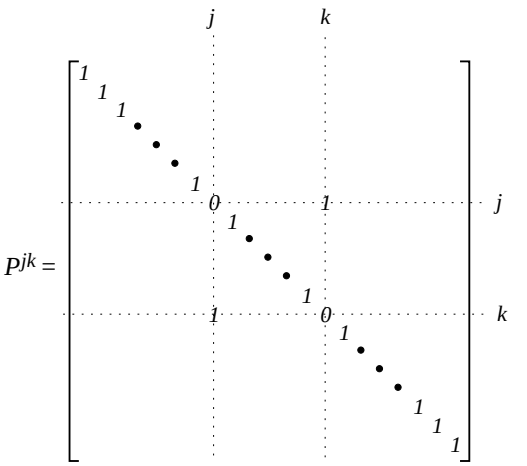


FIGURE 2.2
Elementary permutation matrix $P^{j,k}$

Example 2.4
Repeat Example 2.3 using partial pivoting.

Solution 2.4 The A matrix is repeated here for convenience.

$$A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix}$$

For $j = 1$, the first column of Q is exactly the first column of A. Applying

the pivoting strategy of equation (2.32), the q_{41} element has the largest magnitude of the first column; therefore, rows four and one are exchanged. The elementary permutation matrix $P^{1,4}$ is

$$P^{1,4} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The corresponding A matrix becomes

$$A = \begin{bmatrix} 9 & 2 & 7 & 4 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 1 & 3 & 4 & 8 \end{bmatrix}$$

and Q at the $j = 1$ step:

$$Q = \begin{bmatrix} 9 & \frac{2}{9} & \frac{7}{9} & \frac{4}{9} \\ 2 & & & \\ 4 & & & \\ 1 & & & \end{bmatrix}$$

At $j = 2$, the calculation of the second column of Q yields

$$Q = \begin{bmatrix} 9 & \frac{2}{9} & \frac{7}{9} & \frac{4}{9} \\ 2 & \frac{5}{9} & & \\ 4 & \frac{19}{9} & & \\ 1 & \frac{25}{9} & & \end{bmatrix}$$

Searching the elements in the j^{th} column below the diagonal, the fourth row of the j^{th} (i.e., second) column once again yields the largest magnitude. Therefore rows two and four must be exchanged, yielding the elementary permutation matrix $P^{2,4}$:

$$P^{2,4} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Similarly, the updated A is

$$\begin{bmatrix} 9 & 2 & 7 & 4 \\ 1 & 3 & 4 & 8 \\ 4 & 3 & 5 & 8 \\ 2 & 1 & 2 & 3 \end{bmatrix}$$

which yields the following Q:

$$Q = \begin{bmatrix} 9 & \frac{2}{9} & \frac{7}{9} & \frac{4}{9} \\ 1 & \frac{25}{9} & \frac{29}{25} & \frac{68}{25} \\ 4 & \frac{19}{9} & & \\ 2 & \frac{5}{9} & & \end{bmatrix}$$

For $j = 3$, the calculation of the third column of Q yields:

$$Q = \begin{bmatrix} 9 & \frac{2}{9} & \frac{7}{9} & \frac{4}{9} \\ 1 & \frac{25}{9} & \frac{29}{25} & \frac{68}{25} \\ 4 & \frac{19}{9} & \check{S} & \frac{14}{25} \\ 2 & \frac{5}{9} & \check{S} & \frac{1}{5} \end{bmatrix}$$

In this case, the diagonal element has the largest magnitude, so no pivoting is required. Continuing with the calculation of the 3rd row of Q yields:

$$Q = \begin{bmatrix} 9 & \frac{2}{9} & \frac{7}{9} & \frac{4}{9} \\ 1 & \frac{25}{9} & \frac{29}{25} & \frac{68}{25} \\ 4 & \frac{19}{9} & \check{S} & \check{S} \; \frac{12}{14} \\ 2 & \frac{5}{9} & \check{S} & \frac{1}{5} \end{bmatrix}$$

Lastly, calculating q_{44} yields the "nal Q matrix:

$$Q = \begin{bmatrix} 9 & \frac{2}{9} & \frac{7}{9} & \frac{4}{9} \\ 1 & \frac{25}{9} & \frac{29}{25} & \frac{68}{25} \\ 4 & \frac{19}{9} & \check{S} & \check{S} \; \frac{12}{14} \\ 2 & \frac{5}{9} & \check{S} & \frac{1}{5} \quad \frac{3}{7} \end{bmatrix}$$

The permutation matrix P is found by multiplying together the two elementary permutation matrices:

$$P = P^{2,4} P^{1,4} I$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The results can be checked to verify that $PA = LU$. The forward and backward substitution steps are carried out on the modi"ed vector $b = Pb$.

2.2.2 LU Factorization with Complete Pivoting

An alternate LU factorization that allows complete pivoting is the Gauss• method. In this approach, two permutation matrices are developed: one for row exchange as in partial pivoting, and a second matrix for column exchange. In this approach, the LU factors are found such that

$$P_1 A P_2 = LU \tag{2.33}$$

Therefore to solve the linear system of equations $Ax = b$ requires that a slightly di erent approach be used. As with partial pivoting, the permutation matrix P_1 premultiplies the linear system:

$$P_1 Ax = P_1 b = b \tag{2.34}$$

Now, de“ne a new vector z such that

$$x = P_2 z \tag{2.35}$$

Then substituting equation (2.35) into equation (2.34) yields

$$P_1 A P_2 z = P_1 b = b$$
$$LUz = b \tag{2.36}$$

where equation (2.36) can be solved using forward and backward substitution for z. Once z is obtained, then the solution vector x follows from equation (2.35).

In complete pivoting, both rows and columns may be interchanged to place the largest element (in magnitude) on the diagonal at each step in the LU factorization process. The pivot element is chosen from the remaining elements below and to the right of the diagonal.

Complete Pivoting Strategy

1. At the j^{th} step of LU factorization, choose the pivot element such that

$$|a_{jj}| = \max |a_{kl}| \text{ for } k = j, \ldots, n, \text{ and } l = j, \ldots, n \tag{2.37}$$

2. Exchange rows and update A, P, and Q correspondingly.

Gauss• Algorithm for Computing LU from A

1. Initialize Q to the zero matrix. Let $j = 1$.
2. Set the j^{th} column of Q (j^{th} column of L) to the j^{th} column of the reduced matrix $A^{(j)}$, where $A^{(1)} = A$, and

$$a_{kj} = a_{kj}^{(j)} \text{ for } k = j, \ldots, n \tag{2.38}$$

3. If $j = n$, then stop.
4. Assuming that $a_{jj} = 0$, set the j^{th} row of Q (j^{th} row of U) as

$$a_{jk} = \frac{a_{jk}^{(j)}}{a_{jj}} \text{ for } k = j + 1, \ldots, n \tag{2.39}$$

5. Update $A^{(j+1)}$ from $A^{(j)}$ as

$$a_{ik}^{(j+1)} = a_{ik}^{(j)} - a_{ij}\, a_{jk} \text{ for } i = j + 1, \ldots, n, \text{ and } k = j + 1, \ldots, n \tag{2.40}$$

6. Set $j = j + 1$. Go to step 2.

This factorization algorithm gives rise to the same number of multiplications and divisions as Crout•s algorithm for LU factorization. Crout•s algorithm uses each entry of the A matrix only once, whereas Gauss• algorithm updates the A matrix each time. One advantage of Crout•s algorithm over Gauss• algorithm is each element of the A matrix is used only once. Since each q_{jk} is a function of a_{jk} and then a_{jk} is never used again, the element q_{jk} can be written over the a_{jk} element. Therefore, rather than having to store two $n \times n$ matrices in memory (A and Q), only one matrix is required.

The Crout•s and Gauss• algorithms are only two of numerous algorithms for LU factorization. Other methods include Doolittle and bifactorization algorithms [20], [26], [49]. Most of these algorithms require similar numbers of multiplications and divisions and only di er slightly in performance when implemented on traditional serial computers. However, these algorithms di er considerably when factors such as memory access, storage, and parallelization are considered. Consequently, it is wise to choose the factorization algorithm to "t the application and the computer architecture upon which it will be implemented.

2.3 Condition Numbers and Error Propagation

The Gaussian elimination and LU factorization algorithms are considered direct methods because they calculate the solution vector $x = A^{-1}b$ in a "nite number of steps without an iterative re"nement. On a computer with in"nite precision, direct methods would yield the exact solution x. However, since computers have "nite precision, the solution obtained has limited accuracy. The condition number of a matrix is a useful measure for determining the level of accuracy of a solution. The condition number of the matrix A is generally de"ned as

$$(A) = \frac{\max}{\min} \tag{2.41}$$

where $_{max}$ and $_{min}$ denote the largest and smallest eigenvalues of the matrix $A^T A$. These eigenvalues are real and non-negative regardless of whether the eigenvalues of A are real or complex.

The condition number of a matrix is a measure of the linear independence of the eigenvectors of the matrix. A singular matrix has at least one zero eigenvalue and contains at least one degenerate row (i.e., the row can be expressed as a linear combination of other rows). The identity matrix, which gives rise to the most linearly independent eigenvectors possible and has every eigenvalue equal to one, has a condition number of 1. If the condition number of a matrix is much much greater than one, then the matrix is said to be ill conditioned. The larger the condition number, the more sensitive the solution

process is to slight perturbations in the elements of A and the more numerical error likely to be contained in the solution.

Because of numerical error introduced into the solution process, the computed solution x of equation (2.1) will differ from the exact solution x by a finite amount Δx. Other errors, such as approximation, measurement, or round-off error, may be introduced into the matrix A and vector b. Gaussian elimination produces a solution that has roughly

$$t \log_{10} \beta - \log_{10} \kappa(A) \tag{2.42}$$

correct decimal places in the solution, where t is the bit length of the mantissa ($t = 24$ for a typical 32-bit binary word), β is the base ($\beta = 2$ for binary operations), and κ is the condition number of the matrix A. One interpretation of equation (2.42) is that the solution will lose about $\log_{10} \kappa$ digits of accuracy during Gaussian elimination (and consequently LU factorization). Based upon the known accuracy of the matrix entries, the condition number, and the machine precision, the accuracy of the numerical solution x can be predicted [35].

2.4 Relaxation Methods

Relaxation methods are iterative in nature and produce a sequence of vectors that ideally converge to the solution $x = A^{-1}b$. Relaxation methods can be incorporated into the solution of equation (2.1) in several ways. In all cases, the principal advantage of using a relaxation method stems from not requiring a direct solution of a large system of linear equations and from the fact that the relaxation methods permit the simulator to exploit the latent portions of the system (those portions which are relatively unchanging at the present time) effectively. In addition, with the advent of parallel-processing technology, relaxation methods lend themselves more readily to parallel implementation than do direct methods. The two most common relaxation methods are the Jacobi and the Gauss-Seidel methods [56].

These relaxation methods may be applied for the solution of the linear system

$$Ax = b \tag{2.43}$$

A general approach to relaxation methods is to define a splitting matrix M such that equation (2.43) can be rewritten in equivalent form as

$$Mx = (M - A)x + b \tag{2.44}$$

This splitting leads to the iterative process

$$Mx^{k+1} = (M - A)x^k + b \quad k = 1, \ldots, \tag{2.45}$$

where k is the iteration index. This iteration produces a sequence of vectors $x^1, x^2, \ldots$ for a given initial guess x^0. Various iterative methods can be developed by different choices of the matrix M. The objective of a relaxation method is to choose the splitting matrix M such that the sequence is easily computed and the sequence converges rapidly to a solution.

Let A be split into $L + D + U$, where L is strictly lower triangular, D is a diagonal matrix, and U is strictly upper triangular. Note that these matrices are different from the L and U obtained from LU factorization. The vector x can then be solved for in an iterative manner using the Jacobi relaxation method,

$$x^{k+1} = -D^{-1}(L + U)x^k - b \tag{2.46}$$

or identically in scalar form,

$$x_i^{k+1} = -\sum_{j=i}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}} \qquad 1 \le i \le n, \; k \ge 0 \tag{2.47}$$

In the Jacobi relaxation method, all of the updates of the approximation vector x^{k+1} are obtained by using only the components of the previous approximation vector x^k. Therefore this method is also sometimes called the method of simultaneous displacements.

The Gauss-Seidel relaxation method is similar:

$$x^{k+1} = -(L + D)^{-1} U x^k - b \tag{2.48}$$

or in scalar form

$$x_i^{k+1} = -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{k+1} - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^k + \frac{b_i}{a_{ii}} \qquad 1 \le i \le n, \; k \ge 0 \tag{2.49}$$

The Gauss-Seidel method has the advantage that each new update x_i^{k+1} relies only on previously computed values at that iteration: $x_1^{k+1}, x_2^{k+1}, \ldots, x_{i-1}^{k+1}$. Since the states are updated one-by-one, the new values can be stored in the same locations held by the old values, thus reducing the storage requirements.

Since relaxation methods are iterative, it is essential to determine under what conditions they are guaranteed to converge to the exact solution

$$x = A^{-1}b \tag{2.50}$$

It is well known that a necessary and sufficient condition for the Jacobi relaxation method to converge given any initial guess x_0 is that all eigenvalues of

$$M_J = -D^{-1}(L + U) \tag{2.51}$$

must lie within the unit circle in the complex plane [56]. Similarly, the eigenvalues of

$$M_{GS} = -(L + D)^{-1} U \tag{2.52}$$

must lie within the unit circle in the complex plane for the Gauss-Seidel relaxation algorithm to converge for any initial guess x_0. In practice, these conditions are difficult to confirm. There are several more general conditions that are easily confirmed under which convergence is guaranteed. In particular, if A is strictly diagonally dominant, then both the Jacobi and Gauss-Seidel methods are guaranteed to converge to the exact solution.

The initial vector x_0 can be arbitrary; however if a good guess of the solution is available it should be used for x_0 to produce more rapid convergence to within some pre-defined tolerance.

In general, the Gauss-Seidel method converges faster than the Jacobi for most classes of problems. If A is lower-triangular, the Gauss-Seidel method will converge in one iteration to the exact solution, whereas the Jacobi method will take n iterations. The Jacobi method has the advantage, however, that at each iteration, each x_i^{k+1} is independent of all other x_j^{k+1} for $j = i$. Thus the computation of all x_i^{k+1} can proceed in parallel. This method is therefore well suited to parallel processing [36].

Both the Jacobi and Gauss-Seidel methods can be generalized to the block-Jacobi and block-Gauss-Seidel methods where A is split into block matrices $L + D + U$, where D is block diagonal and L and U are lower- and upper-block triangular respectively. The same necessary and sufficient convergence conditions exist for the block case as for the scalar case, that is, the eigenvalues of M_J and M_{GS} must lie within the unit circle in the complex plane.

Example 2.5

Solve

$$\begin{bmatrix} 10 & -2 & -3 & -6 \\ -0 & 9 & -1 & -4 \\ -2 & -6 & 12 & -2 \\ -3 & -1 & -0 & 8 \end{bmatrix} x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \tag{2.53}$$

for x using (1) the Gauss-Seidel method, and (2) the Jacobi method.

Solution 2.5 The Gauss-Seidel method given in equation (2.49) with the initial vector $x = [0\ 0\ 0\ 0]$ leads to the following updates:

k	x_1	x_2	x_3	x_4
1	0.0000	0.0000	0.0000	0.0000
2	-0.1000	-0.2222	-0.3778	-0.5653
3	-0.5969	-0.5154	-0.7014	-0.7883
4	-0.8865	-0.6505	-0.8544	-0.9137
5	-1.0347	-0.7233	-0.9364	-0.9784
6	-1.1126	-0.7611	-0.9791	-1.0124
7	-1.1534	-0.7809	-1.0014	-1.0301
8	-1.1747	-0.7913	-1.0131	-1.0394
9	-1.1859	-0.7968	-1.0193	-1.0443
10	-1.1917	-0.7996	-1.0225	-1.0468
11	-1.1948	-0.8011	-1.0241	-1.0482
12	-1.1964	-0.8019	-1.0250	-1.0489
13	-1.1972	-0.8023	-1.0255	-1.0492
14	-1.1976	-0.8025	-1.0257	-1.0494
15	-1.1979	-0.8026	-1.0259	-1.0495
16	-1.1980	-0.8027	-1.0259	-1.0496

The Gauss-Seidel iterates have converged to the solution

$$x = [\,\check{S}1.1980 \;\check{S}0.8027 \;\check{S}1.0259 \;\check{S}1.0496\,]^T$$

From equation (2.47) and using the initial vector x = [0 0 0 0], the following updates are obtained for the Jacobi method:

k	x_1	x_2	x_3	x_4
1	0.0000	0.0000	0.0000	0.0000
2	-0.1000	-0.2222	-0.2500	-0.5000
3	-0.5194	-0.4722	-0.4611	-0.5653
4	-0.6719	-0.5247	-0.6669	-0.7538
5	-0.8573	-0.6314	-0.7500	-0.8176
6	-0.9418	-0.6689	-0.8448	-0.9004
7	-1.0275	-0.7163	-0.8915	-0.9368
8	-1.0728	-0.7376	-0.9355	-0.9748
9	-1.1131	-0.7594	-0.9601	-0.9945
10	-1.1366	-0.7709	-0.9810	-1.0123
11	-1.1559	-0.7811	-0.9936	-1.0226
12	-1.1679	-0.7871	-1.0037	-1.0311
13	-1.1772	-0.7920	-1.0100	-1.0363
14	-1.1832	-0.7950	-1.0149	-1.0404
15	-1.1877	-0.7974	-1.0181	-1.0431
16	-1.1908	-0.7989	-1.0205	-1.0451
17	-1.1930	-0.8001	-1.0221	-1.0464
18	-1.1945	-0.8009	-1.0233	-1.0474
19	-1.1956	-0.8014	-1.0241	-1.0480
20	-1.1963	-0.8018	-1.0247	-1.0485
21	-1.1969	-0.8021	-1.0250	-1.0489
22	-1.1972	-0.8023	-1.0253	-1.0491
23	-1.1975	-0.8024	-1.0255	-1.0492
24	-1.1977	-0.8025	-1.0257	-1.0494
25	-1.1978	-0.8026	-1.0258	-1.0494

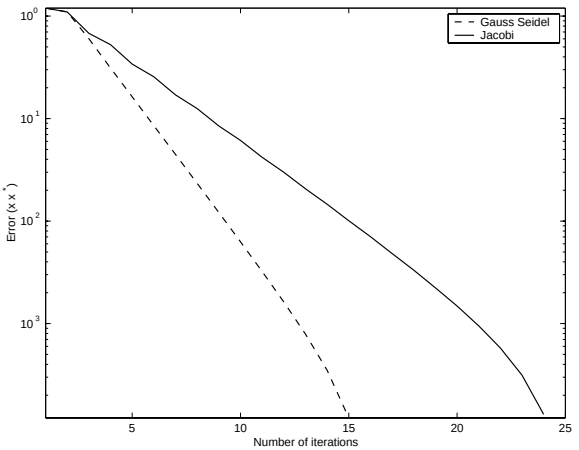


FIGURE 2.3
Convergence rates of the Gauss-Seidel and Jacobi methods

The Jacobi iterates have converged to the same solution as the Gauss-Seidel method. The error in the iterates is shown in Figure 2.3 on a semi-log scale, where the error is de“ned as the maximum $|(x_i^k - x_i)|$ for all $i = 1, \ldots, 4$. Both the Gauss-Seidel and the Jacobi methods exhibit linear convergence, but the Gauss-Seidel converges with a steeper slope and will therefore reach the convergence tolerance sooner for the same initial condition.

Example 2.6
Repeat Example 2.2 using the Jacobi iterative method.

Solution 2.6 Repeating the solution procedure of Example 2.5 yields the following iterations for the Jacobi method:

k	x_1	x_2	x_3	x_4
1	0	0	0	0
2	1.0000	1.0000	0.2000	0.2500
3	-4.8000	-2.1500	-1.6000	-2.8500
4	36.6500	22.3500	9.8900	14.9250
5	-225.0100	-136.8550	-66.4100	-110.6950

Obviously these iterates are not converging. To understand why they are diverging, consider the iterative matrix for the Jacobi matrix:

$$M_J = -D^{-1}(L + U)$$

$$= \begin{bmatrix} 0.00 & -3.00 & -4.00 & -8.00 \\ -2.00 & 0.00 & -2.00 & -3.00 \\ -0.80 & -0.60 & 0.00 & -1.60 \\ -2.25 & -0.50 & -1.75 & 0.00 \end{bmatrix}$$

The eigenvalues of M_J are

$$\begin{bmatrix} -6.6212 \\ 4.3574 \\ 1.2072 \\ 1.0566 \end{bmatrix}$$

which are all greater than one and lie outside the unit circle. Therefore, the Jacobi method will not converge to the solution regardless of choice of initial condition and cannot be used to solve the system of Example 2.2.

If the largest eigenvalue of the iterative matrix M_J or M_{GS} is less than, but almost, unity, then the convergence may proceed very slowly. In this case it is desirable to introduce a weighting factor that will improve the rate of convergence. From

$$x^{k+1} = -(L + D)^{-1} Ux^k - b \tag{2.54}$$

it follows that

$$x^{k+1} = x^k - D^{-1} Lx^{k+1} + (D + U)x^k - b \tag{2.55}$$

A new iterative method can be defined with the weighting factor such that

$$x^{k+1} = x^k - D^{-1} Lx^{k+1} + (D + U)x^k - b \tag{2.56}$$

This method is known as the successive overrelaxation (SOR) method with relaxation coefficient > 0. Note that if the relaxation iterates converge, they converge to the solution $x = A^{-1}b$. One necessary condition for the SOR method to be convergent is that $0 < < 2$ [27]. The calculation of the optimal value for is difficult, except in a few simple cases. The optimal value is usually determined through trial and error, but analysis shows that for systems larger than $n = 30$, the optimal SOR can be more than forty times faster than the Jacobi method [27]. The improvement on the speed of convergence often improves as n increases.

2.5 Conjugate Gradient Methods

Another common iterative method for solving $Ax = b$ is the conjugate gradient method. This method can be considered a minimization method for the function

$$E(x) = \|Ax - b\|^2 \tag{2.57}$$

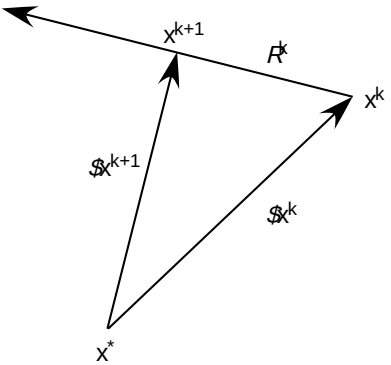


FIGURE 2.4
The conjugate gradient method

along a succession of rays. One attractive feature of this method is that it is guaranteed to converge in at most n steps (neglecting round-off error) if the A matrix is positive definite. The conjugate gradient method is most frequently used instead of Gaussian elimination if the A matrix is very large and sparse, in which case the solution may be obtained in less than n steps. This is especially true if the A matrix is well conditioned. If the matrix is ill conditioned, then round-off errors may prevent the algorithm from obtaining a sufficiently accurate solution after n steps.

In the conjugate gradient method, a succession of search directions p_k is employed and a parameter α_k is computed such that $f(x^k + \alpha_k p_k)$ is minimized along the p_k direction. Upon setting x^{k+1} equal to $x^k + \alpha_k p_k$, the new search direction is found. As the conjugate gradient method progresses, each error function is associated with a specific ray, or orthogonal expansion. Therefore the conjugate gradient method is reduced to the process of generating the orthogonal vectors and finding the proper coefficients to represent the desired solution. The conjugate gradient method is illustrated in Figure 2.4. Let x denote the exact (but unknown) solution, x^k an approximate solution, and $\delta x^k = x^k - x$. Given any search direction p^k, the minimal distance from the line to x is found by constructing x^{k+1} perpendicular to x^k. Since the exact solution is unknown, the residual is made to be perpendicular to p^k. Regardless of how the new search direction is chosen, the norm of the residual will not increase.

All iterative methods for solving $Ax = b$ define an iterative process such that

$$x^{k+1} = x^k + \alpha_{k+1} p_{k+1} \tag{2.58}$$

where x^{k+1} is the updated value, α_k is the steplength, and p_k defines the direction R^n in which the algorithm moves to update the estimate.

Let the residual, or mismatch, vector at step k be given by

$$r_k = Ax^k - b \tag{2.59}$$

and the error function given by

$$E_k(x^k) = \|Ax^k - b\|^2 \tag{2.60}$$

Then the coefficient that minimizes the error function at step $k+1$ is

$$\alpha_{k+1} = \frac{\|A^T r_k\|^2}{\|Ap_{k+1}\|^2} \tag{2.61}$$

This has the geometric interpretation of minimizing E_{k+1} along the ray defined by p_{k+1}. Further, an improved algorithm is one that seeks the minimum of E_{k+1} in a plane spanned by two direction vectors, such that

$$x^{k+1} = x^k + \alpha_{k+1}(p_{k+1} + \beta_{k+1}p_{k+1}) \tag{2.62}$$

where the rays p_{k+1} and p_{k+1} span a plane in R^n. The process of selecting direction vectors and coefficients to minimize the error function E_{k+1} is optimized when the chosen vectors are orthogonal, such that

$$\langle Ap_{k+1}, Ap_{k+1}\rangle = 0 \tag{2.63}$$

where $\langle\cdot\rangle$ denotes inner product. Vectors that satisfy the orthogonality condition of equation (2.63) are said to be mutually conjugate with respect to the operator A^TA, where A^T is the conjugate transpose of A. One method of choosing appropriate vectors is to choose p_{k+1} as a vector orthogonal to p_k, thus eliminating the need to specify two orthogonal vectors at each step. While this simplifies the procedure, there is now an implicit recursive dependence for generating the p vectors.

Conjugate Gradient Algorithm for Solving $Ax = b$

Initialization: Let $k = 0$, and

$$r_0 = Ax^0 - b \tag{2.64}$$

$$p_0 = -A^T r_0 \tag{2.65}$$

While r_k

$$\alpha_{k+1} = \frac{\|A^T r_k\|^2}{\|Ap_k\|^2} \tag{2.66}$$

$$x^{k+1} = x^k + \alpha_{k+1}p_k \tag{2.67}$$

$$r_{k+1} = Ax^{k+1} - b \tag{2.68}$$

$$B_{k+1} = \frac{\|A^T r_{k+1}\|^2}{\|A^T r_k\|^2} \tag{2.69}$$

$$p_{k+1} = -A^T r_{k+1} + B_{k+1}p_k \tag{2.70}$$

$$k = k + 1 \tag{2.71}$$

For an arbitrary nonsingular positive definite matrix A, the conjugate gradient method will produce a solution in at most n steps (neglecting round-off error). This is a direct consequence of the fact that then direction vectors $p_0, p_1, \ldots$ span the solution space. Finite step termination is a significant advantage of the conjugate gradient method over other iterative methods such as relaxation methods.

Example 2.7
Repeat Example 2.5 using the conjugate gradient method.

Solution 2.7 The problem of Example 2.5 is repeated here for convenience: Solve

$$\begin{bmatrix} 10 & 2 & 3 & 6 \\ 0 & 9 & 1 & 4 \\ 2 & 6 & 12 & 2 \\ 3 & 1 & 0 & 8 \end{bmatrix} x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \tag{2.72}$$

with $x^0 = [0\ 0\ 0\ 0]^T$.

Initialization: Let $k = 1$, and

$$r_0 = Ax^0 - b = \begin{bmatrix} -1 \\ -2 \\ -3 \\ -4 \end{bmatrix} \tag{2.73}$$

$$p_0 = -A^T r_0 = \begin{bmatrix} 8 \\ 6 \\ -31 \\ -12 \end{bmatrix} \tag{2.74}$$

The initial error is

$$E_0 = \|r_0\|^2 = (5.4772)^2 = 30 \tag{2.75}$$

Iteration 1

$$\alpha_0 = \frac{\|A^T r_0\|^2}{\|Ap_0\|^2} = 0.0049 \tag{2.76}$$

$$x^1 = x^0 + \alpha_0 p_0 = \begin{bmatrix} 0.0389 \\ 0.0292 \\ -0.1507 \\ -0.0583 \end{bmatrix} \tag{2.77}$$

$$r_1 = Ax^1 - b = \begin{bmatrix} -2.1328 \\ -2.6466 \\ -1.0553 \\ -3.3874 \end{bmatrix} \tag{2.78}$$

$$B_1 = \frac{\|A^T r_1\|^2}{\|A^T r_0\|^2} = 0.1613 \tag{2.79}$$

$$p_1 = \check{S}A^T r_1 + B_1 p_0 = \begin{matrix} \check{S}7.7644 \\ \check{S}8.8668 \\ \check{S}8.6195 \\ \check{S}3.5414 \end{matrix} \tag{2.80}$$

and the error is

$$E_1 = \|r_1\|^2 = (4.9134)^2 = 24.1416 \tag{2.81}$$

Similarly for iterations 2…4:

k	α_k	x^k	r_k	B_k	p_k	$\|r_k\|$
		Š0.3211	Š1.5391		Š24.9948	
2	0.0464	Š0.3820	Š0.0030	2.5562	Š17.4017	3.8895
		Š0.5504	0.2254		Š14.7079	
		Š0.2225	Š3.5649		Š28.7765	
		Š0.9133	Š1.5779		Š33.5195	
3	0.0237	Š0.7942	Š0.6320	0.7949	Š1.0021	1.8322
		Š0.8988	Š0.6146		Š14.9648	
		Š0.9043	Š0.2998		Š17.1040	
		Š1.1981	0.0000		0.0000	
4	0.0085	Š0.8027	0.0000	0.0000	0.0000	0.0000
		Š1.0260	0.0000		0.0000	
		Š1.0496	0.0000		0.0000	

The iterations converged in four steps as the algorithm guarantees.

Unfortunately for a general linear system, the conjugate gradient method requires signi“cantly more multiplications and divisions than does the LU factorization method. The conjugate gradient method is more numerically competitive for matrices that are very large and sparse or that have a special structure that cannot be easily handled by LU factorization. In some cases, the speed of convergence of the conjugate gradient method can be improved by preconditioning.

As seen with the Gauss-Seidel and Jacobi iteration, the convergence rate of iterative algorithms is closely related to the eigenvalue spectrum of the iterative matrix. Consequently, scaling or matrix transformation that converts the original system of equations into one with a better eigenvalue spectrum might signi“cantly improve the rate of convergence. This procedure is known as preconditioning. A number of systematic approaches for sparse-matrix preconditioning have been developed in which the basic approach is to convert the system

$$Ax = b$$

into an equivalent system

$$M^{\check{S}1}Ax = M^{\check{S}1}b$$

where M^{-1} approximates A^{-1}. For example, one such approach might be to use an incomplete LU factorization, where the LU factorization method is applied but all "lls are neglected. If the A matrix is diagonally dominant, a simple approximation to M^{-1} is

$$M^{-1} = \begin{bmatrix} \frac{1}{A(1,1)} & & & \\ & \frac{1}{A(2,2)} & & \\ & & \ddots & \\ & & & \frac{1}{A(n,n)} \end{bmatrix} \tag{2.82}$$

This preconditioning strategy scales the system of equations so that the entries along the main diagonal are all equal. This procedure can compensate for orders-of-magnitude di erences in scale. Note that scaling will not have any e ect on matrices that are inherently ill conditioned.

Example 2.8
Repeat Example 2.7 using a preconditioner.

Solution 2.8 Let M^{-1} be de"ned as in equation (2.82). Thus

$$M^{-1} = \begin{bmatrix} \frac{1}{10} & & & \\ & \frac{1}{9} & & \\ & & \frac{1}{12} & \\ & & & \frac{1}{8} \end{bmatrix} \tag{2.83}$$

$$A = M^{-1}A = \begin{bmatrix} 1.0000 & -0.2000 & -0.3000 & -0.6000 \\ 0.0000 & 1.0000 & -0.1111 & -0.4444 \\ -0.1667 & -0.5000 & 1.0000 & -0.1667 \\ -0.3750 & -0.1250 & 0.0000 & 1.0000 \end{bmatrix} \tag{2.84}$$

$$b = M^{-1}b = \begin{bmatrix} -0.1000 \\ -0.2222 \\ -0.2500 \\ -0.5000 \end{bmatrix} \tag{2.85}$$

Solving $A\,x = b$ using the conjugate gradient method yields the following set of errors:

k	E_k^2
0	0.6098
1	0.5500
2	0.3559
3	0.1131
4	0.0000

Although it takes the same number of iterations to converge, note that the errors for $k < 4$ are much smaller than in Example 2.7. For large systems, it is

conceivable that the error would be decreased sufficiently rapidly to terminate the iterations prior to the n-th step. This method is also useful if only an approximate solution to x is desired.

2.6 Generalized Minimal Residual Algorithm (GMRES)

If the matrix A is neither symmetric nor positive definite, then the term

$$A_{k+1}, A_{k+1}$$

is not guaranteed to be zero and the search vectors are not mutually orthogonal. Mutual orthogonality is required to generate a basis of the solution space. Hence this basis must be explicitly constructed. The extension of the conjugate gradient method, called the Generalized Minimal Residual Algorithm (GMRES), minimizes the norm of the residual in a subspace spanned by the set of vectors

$$r^0, Ar^0, A^2r^0, \ldots, A^{k-1}r^0$$

where vector r^0 is the initial residual $r^0 = b - Ax^0$, and the k-th approximation to the solution is chosen from this space. This subspace, called Krylov subspace, is made orthogonal by the well-known Gram-Schmidt procedure, known as the Arnoldi process when applied to a Krylov subspace [37]. At each step k, the GMRES algorithm applies the Arnoldi process to a set of k orthonormal basis vectors for the kth Krylov subspace to generate the next basis vector. Arnoldi methods are described in greater detail in Section 7.3. At each step, the algorithm multiplies the previous Arnoldi vector v_j by A and then orthonormalizes the resulting vector w_j against all previous v_i's. The columns $V = [v_1, v_2, \ldots, v_k]$ form an orthonormal basis for the Krylov subspace and H is the orthogonal projection of A onto this space.

An orthogonal matrix triangularization such as the Arnoldi method consists in determining an $n \times n$ orthogonal matrix Q such that

$$Q^T = \begin{matrix} R \\ 0 \end{matrix} \tag{2.86}$$

where R is an $m \times m$ upper triangular matrix R. Then the solution process reduces to solving the triangular system $Rx = Py$, where P consists of the first m rows of Q.

To clear one element at a time to upper triangularize a matrix, the Given's rotation can be applied. The Given's rotation is a transformation based on

the matrix

$$\begin{matrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & & & \vdots \\ 0 & \cdots & cs & \cdots & sn & \cdots & 0 \\ \vdots & & \vdots & \ddots & & & \vdots \\ 0 & \cdots & -sn & \cdots & cs & \cdots & 0 \\ \vdots & & \vdots & & & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{matrix} \qquad (2.87)$$

with properly chosen $cn = \cos(\ )$ and $sn = \sin(\ )$ for some rotation angle can be used to zero the element A_{ki}.

One of the difficulties with the GMRES methods is that as k increases, the number of vectors requiring storage increases as k and the number of multiplications as $\frac{1}{2}k^2n$ (for a $n \times n$ matrix). To remedy this difficulty, the algorithm can be applied iteratively, i.e. it can be restarted every m steps, where m is some "fixed integer parameter. This is often called the GMRES(m) algorithm.

GMRES(m) Algorithm for Solving $Ax = b$

Initialization:

Let $k = 0$, and

$$r_0 = Ax^0 - b$$
$$e_1 = [1\ 0\ 0\ \ldots\ 0]^T$$

While r_k and $k < k_{max}$, set

$$j = 1$$
$$v_1 = r/\|r\|$$
$$s = \|r\| e_1$$

$$cs = [0\ 0\ 0\ \ldots\ 0]^T$$
$$sn = [0\ 0\ 0\ \ldots\ 0]^T$$

While $j < m$,

1. Arnoldi process

   (a) form the matrix H such that $H(i,j) = (Av_j)^T v_i$, $i = 1, \ldots, j$

   (b) Let $w = Av_j - \sum_{i=1}^{j} H(i,j)v_i$

   (c) Set $H(j+1, j) = \|w\|$

   (d) Set $v_{j+1} = w/\|w\|$

2. Givens rotation

   (a) Compute

$$\begin{matrix} H(i,j) \\ H(i+1,j) \end{matrix} = \begin{matrix} cs(i) & sn(i) \\ -sn(i) & cs(i) \end{matrix} \begin{matrix} H(i,j) \\ H(i+1,j) \end{matrix}$$

   $i = 1, \ldots, j-1$

   (b) Set

$$cs(j) = \frac{H(j,j)}{H(j+1,j)^2 + H(j,j)^2}$$

$$sn(j) = \frac{H(j+1,j)}{H(j+1,j)^2 + H(j,j)^2}$$

   (c) Approximate residual norm

$$\begin{aligned} s(j) &= cs(j)s(j) \\ s(j+1) &= -sn(j)s(j) \\ \text{error} &= |s(j+1)| \end{aligned}$$

   (d) Set

$$\begin{aligned} H(j,j) &= cs(j)H(j,j) + sn(j)H(j+1,j) \\ H(j+1,j) &= 0 \end{aligned}$$

3. If error

   (a) Solve $Hy = s$ for y

   (b) Update approximation

$$x = x - Vy$$

   (c) Method has converged. Return.

4. Set $j = j + 1$

5. If $j = m$ and error > (restart GMRES)

   (a) Set $s = [1\ 0\ 0\ \ldots\ 0]$

   (b) Solve $Hy = s$ for y

   (c) Calculate $x = x - Vy$

   (d) Calculate $r = Ax - b$

   (e) Set $k = k + 1$

Example 2.9
Repeat Example 2.5 using the GMRES method.

Solution 2.9 The problem of Example 2.5 is repeated here for convenience: Solve

$$\begin{bmatrix} 10 & 2 & 3 & 6 \\ 0 & 9 & 1 & 4 \\ 2 & 6 & 12 & 2 \\ 3 & 1 & 0 & 8 \end{bmatrix} x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \qquad (2.88)$$

with $x^0 = [0\ 0\ 0\ 0]^T$. Let $= 10^{-3}$.

j=1 Solving the Arnoldi process yields

$$H = \begin{bmatrix} 4.0333 & 0 & 0 & 0 \\ 6.2369 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.1826 & 0.9084 & 0 & 0 \\ 0.3651 & 0.2654 & 0 & 0 \\ 0.5477 & 00556 & 0 & 0 \\ 0.7303 & 03181 & 0 & 0 \end{bmatrix}$$

Applying the Given•s rotation yields

$$cs = \begin{bmatrix} 0.5430 & 0 & 0 & 0 \end{bmatrix}$$

$$sn = \begin{bmatrix} 0.8397 & 0 & 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 7.4274 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$s = \begin{bmatrix} 2.9743 & 4.5993 & 0 & 0 \end{bmatrix}^T$$

Since error (= |s(2)| = 4 .5993) is greater than , j = j + 1 and repeat.

j=2 Solving the Arnoldi process yields

$$H = \begin{bmatrix} 7.4274 & 26293 & 0 & 0 \\ 0 & 12.5947 & 0 & 0 \\ 0 & 1.9321 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} 0.1826 & 0.9084 & 0.1721 & 0 \\ 0.3651 & 0.2654 & 06905 & 0 \\ 0.5477 & 00556 & 0.6728 & 0 \\ 0.7303 & 03181 & 02024 & 0 \end{bmatrix}$$

Applying the Given•s rotation yields

$$cs = \begin{bmatrix} -0.5430 & 09229 & 0 & 0 \end{bmatrix}$$

$$sn = \begin{bmatrix} 0.8397 & 03850 & 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 7.4274 & -12.0037 & 0 & 0 \\ 0 & 5.0183 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$s = \begin{bmatrix} -2.9743 & -4.2447 & 1.7708 & 0 \end{bmatrix}^T$$

Since error (= |s(3)| = 1.7708) is greater than , j = j + 1 and repeat.

j=3 Solving the Arnoldi process yields

$$H = \begin{bmatrix} 7.4274 & -12.0037 & -3.8697 & 0 \\ 0 & 5.0183 & -0.2507 & 0 \\ 0 & 0 & -13.1444 & 0 \\ 0 & 0 & 2.6872 & 0 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.1826 & -0.9084 & -0.1721 & -0.3343 \\ -0.3651 & -0.2654 & 06905 & 05652 \\ -0.5477 & 00556 & -0.6728 & 04942 \\ -0.7303 & 03181 & 0.2024 & -0.5697 \end{bmatrix}$$

Applying the Given•s rotation yields

$$cs = \begin{bmatrix} -0.5430 & 09229 & -0.9806 & 0 \end{bmatrix}$$

$$sn = \begin{bmatrix} 0.8397 & 03850 & 01961 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 7.4274 & -12.0037 & 18908 & 0 \\ 0 & 5.0183 & -1.9362 & 0 \\ 0 & 0 & 137007 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$s = \begin{bmatrix} -2.9743 & -4.2447 & -1.7364 & -0.3473 \end{bmatrix}^T$$

Since error (= |s(4)| = 0.3473) is greater than , j = j + 1 and repeat.

j=4 Solving the Arnoldi process yields

$$H = \begin{bmatrix} 7.4274 & -12.0037 & 18908 & 14182 \\ 0 & 5.0183 & -1.9362 & 05863 \\ 0 & 0 & 137007 & -1.4228 \\ 0 & 0 & 0 & -9.2276 \end{bmatrix}$$

$$V = \begin{bmatrix} -0.1826 & -0.9084 & -0.1721 & -0.3343 & 07404 \\ -0.3651 & -0.2654 & 06905 & 05652 & 02468 \\ -0.5477 & 0.0556 & -0.6728 & 0.4942 & 0.6032 \\ -0.7303 & 03181 & 02024 & -0.5697 & 01645 \end{bmatrix}$$

Applying the Given•s rotation yields

$$cs = \begin{matrix} \check{S}0.5430 & 09229 & \check{S}0.9806 & 10000 \end{matrix}$$

$$sn = \begin{matrix} 0.8397 & 03850 & 01961 & 00000 \end{matrix}$$

$$H = \begin{matrix} 7.4274 & \check{S}12.0037 & 18908 & \check{S}0.2778 \\ 0 & 5.0183 & \check{S}1.9362 & \check{S}1.9407 \\ 0 & 0 & 137007 & \check{S}1.0920 \\ 0 & 0 & 0 & 91919 \end{matrix}$$

$$s = \begin{matrix} \check{S}2.9743 & \check{S}4.2447 & \check{S}1.7364 & \check{S}0.3473 & 00000 \end{matrix}^T$$

Since error (= |s(5)| = 0), the iteration has converged.

Solving for y from Hy = s yields

$$y = \begin{matrix} \check{S}1.8404 \\ \check{S}0.9105 \\ \check{S}0.1297 \\ \check{S}0.0378 \end{matrix} \tag{2.89}$$

Solving for x from

$$x = x \check{S} V y \tag{2.90}$$

yields

$$x = \begin{matrix} \check{S}1.1981 \\ \check{S}0.8027 \\ \check{S}1.0260 \\ \check{S}1.0496 \end{matrix} \tag{2.91}$$

which is the same as the previous example.

2.7 Problems

1. Show that the number of multiplications and divisions required in the LU factorization of an $n \times n$ square matrix is $n(n^2 - 1)/3$.

2. Consider the system $Ax = b$, where

$$a_{ij} = \frac{1}{i + j - 1} \quad i, j = 1, \ldots, 4$$

and

$$b_i = \frac{1}{3} \sum_{j=1}^{4} a_{ij}$$

Using only four decimal places of accuracy, solve this system using LU factorization

(a) no pivoting

(b) partial pivoting

Comment on the differences in solutions (if any).

3. Prove that the matrix

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

does not have an LU factorization.

4. Assuming that an LU factorization of A is available, write an algorithm to solve the equation $x^T A = b^T$.

5. For the following matrix, find $A = LU$ (no pivoting) and $PA = LU$ (with partial pivoting)

(a)

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 4 & 10 \\ 3 & -13 & 3 & 3 \\ -6 & 4 & 2 & -18 \end{bmatrix}$$

(b)

$$A = \begin{bmatrix} -2 & 1 & 2 & 5 \\ 2 & -1 & 4 & 1 \\ 1 & 4 & -3 & 2 \\ 8 & 2 & 3 & -6 \end{bmatrix}$$

6. Write an LU factorization-based algorithm to “nd the inverse of any nonsingular matrix A.

7. Solve the system of problem 5(b) with the vector

$$b = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \end{matrix}$$

   (a) Using LU factorization and forward/backward substitution
   (b) Using a Gauss-Jacobi iteration. How many iterations are required?
   (c) Using a Gauss-Seidel iteration. How many iterations are required?
   (d) Using the Conjugate Gradient method. How many iterations are required?
   (e) Using the GMRES method. How many iterations are required?

   Use a starting vector of

$$x = \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix}$$

   and a convergence error of 10^{-5} for the iterative methods.

8. Apply the Gauss-Seidel iteration to the system

$$A = \begin{matrix} 0.96326 & 081321 \\ 0.81321 & 068654 \end{matrix}$$

$$b = \begin{matrix} 0.88824 \\ 0.74988 \end{matrix}$$

   Use $x^0 = [0.33116 \;\; 070000]^T$ and explain what happens.

9. Solve the system of equations in problem 2 using the conjugate gradient method.

10. Solve the system of equations in problem 2 using the GMRES method.

11. Consider ann × n tridiagonal matrix of the form

$$T_a = \begin{matrix} a & -1 & & & & \\ -1 & a & -1 & & & \\ & -1 & a & -1 & & \\ & & -1 & a & -1 & \\ & & & -1 & a & -1 \\ & & & & -1 & a \end{matrix}$$

where a is a real number.

(a) Verify that the eigenvalues of T_a are given by

$$\lambda_j = a - 2\cos(\theta_j) \quad j = 1, \ldots, n$$

where

$$\theta = \frac{\pi}{n+1}$$

(b) Let $a = 2$.

i. Will the Jacobi iteration converge for this matrix?

ii. Will the Gauss-Seidel iteration converge for this matrix?

12. An alternative conjugate gradient algorithm for solving $Ax = b$ may be based on the error functional $E_k(x^k) = (x^k - x, x^k - x)$ where $(\cdot)$ denotes inner product. The solution is given as

$$x^{k+1} = x^k + \alpha_k p_k$$

Using $p_1 = -A^T r_0$ and $p_{k+1} = -A^T r_k + \beta_k p_k$, derive this conjugate gradient algorithm. The coefficients α_k and β_k can be expressed as

$$\alpha_{k+1} = \frac{\|r_k\|^2}{\|Ap_{k+1}\|^2}$$

$$\beta_{k+1} = \frac{\|r_{k+1}\|^2}{\|r_k\|^2}$$

Repeat Example 2.7 using this conjugate gradient algorithm.

13. Write a subroutine with two inputs (A, flag) that will generate for any non-singular matrix A, the outputs (Q, P) such that if

- flag=0, $A = LU, P = I$
- flag=1, $PA = LU$

where

$$L = \begin{bmatrix} l_{11} & 0 & 0 & \ldots & 0 \\ l_{21} & l_{22} & 0 & \ldots & 0 \\ l_{31} & l_{32} & l_{33} & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \ldots & l_{nn} \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 1 & u_{12} & u_{13} & \ldots & u_{1n} \\ 0 & 1 & u_{23} & \ldots & u_{2n} \\ 0 & 0 & 1 & \ldots & u_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{bmatrix}$$

and

$$Q = L + U - I$$

14. For the following non-singular matrices, use the subroutine of Problem 13 and obtain matrices P and Q in each of the following cases:

(a)

$$\begin{matrix} 0 & 0 & 1 \\ 3 & 1 & 4 \\ 2 & 1 & 0 \end{matrix}$$

(b)

$$\begin{matrix} 10^{-10} & 0 & 0 & 1 \\ 0 & 0 & 1 & 4 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{matrix}$$

15. Write a subroutine with two inputs (A, b) that will generator for any non-singular matrix A, the output (x) such that

$$Ax = b$$

using forward and backward substitution. This subroutine should incorporate the subroutine developed in Problem 13.

16. Using the subroutines of Problems 13 and 15, solve the following system of equations

$$\begin{matrix} 2 & 5 & 6 & 11 \\ 4 & 6 & 8 & 2 \\ 4 & 3 & 7 & 0 \\ 1 & 26 & 3 & 4 \end{matrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix} = \begin{matrix} 1 \\ 1 \\ 1 \\ 1 \end{matrix}$$

3

Systems of Nonlinear Equations

Many systems can be modeled generically as

$$F(x) = 0 \tag{3.1}$$

where x is an n-vector and F represents a nonlinear mapping with both its domain and range in the n-dimensional real linear space R^n. The mapping F can also be interpreted as being an n-vector of functions

$$F(x) = \begin{matrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{matrix} = 0 \tag{3.2}$$

where at least one of the functions is nonlinear. Each function may or may not involve all n states x_i, but it is assumed that every state appears at least once in the set of functions. The solution x of the nonlinear system cannot, in general, be expressed in closed form. Thus nonlinear systems are usually solved numerically. In many cases, it is possible to "nd an approximate solution x arbitrarily close to the actual solution x, by replacing each approximation with successively better (more accurate) approximations until

$$F(x) \quad 0.$$

Such methods are usually *iterative*. An iterative solution is one in which an initial guess (x^0) to the solution is used to create a sequence x^0, x^1, x^2, ... that (hopefully) converges arbitrarily close to the desired solution x.

Three principal issues arise with the use of iterative methods, namely

1. Is the iterative process well de"ned? That is, can it be successively applied without numerical di culties?
2. Do the iterates (i.e., the sequence of updates) converge to a solution of equation (3.1)? Is the solution the desired solution?
3. How economical is the entire solution process?

The complete (or partial) answers to these issues are enough to "ll several volumes, and as such cannot be discussed in complete detail in this chapter.

These issues, however, are central to the solution of nonlinear systems and cannot be fully ignored. Therefore this chapter will endeavor to provide sufficient detail for the reader to be aware of the advantages (and disadvantages) of different types of iterative methods without providing exhaustive coverage.

3.1 Fixed Point Iteration

Solving a system of nonlinear equations is a complex problem. To better understand the mechanisms involved in a large-scale system, it is instructive to first consider the one dimensional, or scalar, nonlinear system

$$f(x) = 0. \tag{3.3}$$

One approach to solving any nonlinear equation is the tried-and-true "trial and error" method that most engineering and science students have used at one time or another in their careers.

Example 3.1
Find the solution to

$$f(x) = x^2 - 5x + 4 = 0 \tag{3.4}$$

Solution 3.1 This is a quadratic equation that has a closed form solution. The two solutions are

$$x_1, x_2 = \frac{5 \pm \sqrt{(-5)^2 - (4)(4)}}{2} = 1, \ 4$$

If a closed form solution did not exist, however, one approach would be to use a trial and error approach. Since the solution occurs when $f(x) = 0$, the value of $f(x)$ can be monitored and used to refine the estimates to x.

k	x	f (x)
0	0	$0 - 0 + 4 = 4 > 0$
1	2	$4 - 10 + 4 = -2 < 0$
2	0.5	$0.25 - 2.5 + 4 = 1.75 > 0$
3	1.5	$2.25 - 7.5 + 4 = -1.25 < 0$

By noting the sign of the function and whether or not it changes sign, the interval in which the solution lies can be successively narrowed. If a function $f(x)$ is continuous and $f(a) \cdot f(b) < 0$, then the equation $f(x) = 0$ has at least one solution in the interval (a, b). Since $f(0.5) > 0$ and $f(1.5) < 0$ it can be concluded that one of the solutions lies in the interval $(0.5, 1.5)$.

This process, however, tends to be tedious and there is no guidance to determine what the next guess should be other than the bounds established by the change in sign of $f(x)$. A better method would be to write the sequence of updates in terms of the previous guesses. Thus, an iterative function can be defined as:

$$I: \quad x^{k+1} = g\left(x^k\right), \quad k = 1, \ldots, \tag{3.5}$$

This is known as a fixed-point iteration because at the solution

$$x = g(x) \tag{3.6}$$

Example 3.2
Find the solution to equation (3.4) using a fixed point iteration.

Solution 3.2 Equation (3.4) can be rewritten as

$$x = \frac{x^2 + 4}{5} \tag{3.7}$$

Adopting the notation of equation (3.5), the iterative function becomes

$$x^{k+1} = g\left(x^k\right) = \frac{\left(x^k\right)^2 + 4}{5} \tag{3.8}$$

Using this iterative function, the estimates to x are:

k	x^k	$g\left(x^k\right)$
0	0	$\frac{0+4}{5} = 0.8$
1	0.8	$\frac{0.64+4}{5} = 0.928$
2	0.928	$\frac{0.856+4}{5} = 0.971$
3	0.971	$\frac{0.943+4}{5} = 0.989$

It is obvious that this sequence is converging to the solution $x = 1$.

Now consider the same example, except with a different initial guess:

k	x^k	$g\left(x^k\right)$
0	5	$\frac{25+4}{5} = 5.8$
1	5.8	$\frac{33.64+4}{5} = 7.528$
2	7.528	$\frac{56.67+4}{5} = 12.134$

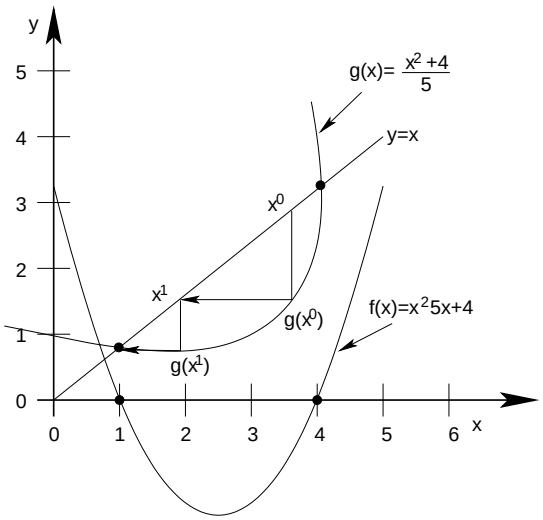


FIGURE 3.1
Graphical interpretation of the "xed point iteration

In this case, the iterates are increasing rapidly and after a few more iterations would approach in"nity. In this case, it is said that the iteration is diverging.

This example brings up two very important points: will a sequence of iterates converge and, if so, to what solution will they converge? In order to address these questions, consider "rst a graphical interpretation of Example 3.2. Plotting both sides of the function in equation (3.7) yields the two lines shown in Figure 3.1.

These two lines intersect at the same two points in which the original function $f(x) = 0$. The "xed point iteration works by "nding this intersection. Consider the initial guess x^0 shown in Figure 3.1. The function $g(x)$ evaluated at x^0 gives the updated iterate x^1. Thus a vertical line projected from x^0 points to $g(x^0)$ and a horizontal line projected from $g(x^0)$ gives x^1.

The projection of the function $g(x^1)$ yields x^2. Similar vertical and horizontal projections will eventually lead directly to the point at which the two lines intersect. In this way, the solution to the original function $f(x)$ can be obtained.

In this example, the solution $x = 1$ is the point of attraction of the "xed point iteration. A point x is said to be a point of attraction of an iterative function I if there exists an open neighborhood S_0 of x such that for all

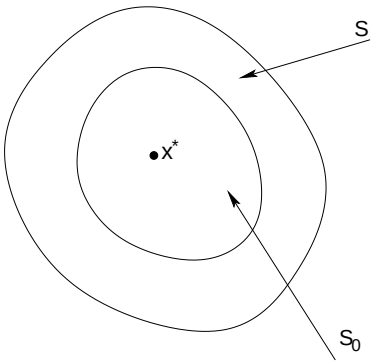


FIGURE 3.2
Domain of attraction of x^*

initial guesses x^0 in the subset S_0 of S, the iterates will remain in S and

$$\lim_k x^k = x^* \tag{3.9}$$

The neighborhood S_0 is called the domain of attraction of x^* [34]. This concept is illustrated in Figure 3.2 and implies that the iterates of I will converge to x^* whenever x^0 is sufficiently close to x^*. In Example 3.2, the fixed point $x^* = 1$ is a point of attraction of

$$I: \quad x^{k+1} = \frac{(x^k)^2 + 4}{5}$$

whereas $x^* = 4$ is not. The domain of attraction of $x^* = 1$ is all x in the domain $-\infty < x < 4$.

It is often difficult to determine *a priori* whether or not an iteration will converge. In some cases, a series of iterates will appear to be converging, but will not approach x^* even as $k \to \infty$. However, there are a number of theorems that provide insight as to whether an iteration of $x = g(x)$ will converge.

Mean Value Theorem: [47] Suppose a function $g(x)$ and its derivative $g'(x)$ are both continuous in the interval $a \le x \le b$. Then there exists at least one ξ, $a < \xi < b$ such that

$$g'(\xi) = \frac{g(b) - g(a)}{b - a}. \tag{3.10}$$

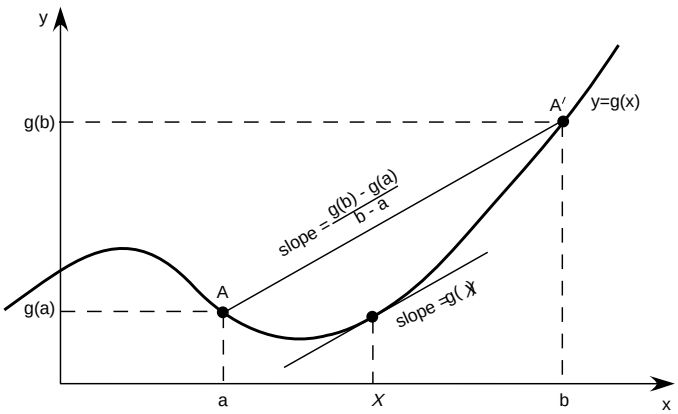


FIGURE 3.3
Meaning of the mean value theorem

The meaning of this theorem is shown in Figure 3.3. If a function $g(x)$ is de“ned in the region between $x = a$ and $x = b$ and is both di erentiable (smooth) and continuous, then a secant line can be drawn between points A and A . The slope of this secant line is

$$\frac{g(b) - g(a)}{b - a}$$

The mean value theorem states that there is at least one point on the curve, at $x =$ where the tangent to the curve has the same slope as the line A .

Rewriting the equation of the mean value theorem as

$$g(b) - g(a) = g'( )(b - a)$$

then for an successive iterates in which $x^{k+1} = b$ and $x^k = a$, then

$$g(x^{k+1}) - g(x^k) = g'(^k)(x^{k+1} - x^k) \tag{3.11}$$

or taking the absoluate values:

$$g(x^{k+1}) - g(x^k) = g'(^k) \quad (x^{k+1} - x^k) \tag{3.12}$$

As long as the appropriate ^{k} is used, the mean value theorem can be successively applied to each iteration of the sequence. If the entire region which includes x as well as all of the x^k, then the derivative $g'(x)$ is bounded. Therefore

$$g'(^k) \quad M \tag{3.13}$$

for any k where M is the positive upper bound. Then starting from the initial guess x^0,

$$x^2 - x^1 \leq M\,|x^1 - x^0| \tag{3.14}$$
$$x^3 - x^2 \leq M\,|x^2 - x^1| \tag{3.15}$$
$$\vdots \tag{3.16}$$
$$x^{k+1} - x^k \leq M\,|x^k - x^{k-1}| \tag{3.17}$$

and by combining yields

$$x^{k+1} - x^k \leq M^k\,|x^1 - x^0| \tag{3.18}$$

Thus for any initial guess x^0,

$$|g'(x)| \leq M < 1 \tag{3.19}$$

then the iterates will converge.

A similar, but slightly different method of determining whether or not an iterative process will converge is given by the Ostrowski theorem [34]. This theorem states that if the iterative process

$$I: \quad x^{k+1} = g(x^k), \quad k = 1, \ldots,$$

has a fixed point x and is continuous and differentiable at x, and if $\frac{\partial g(x)}{\partial x} < 1$, then x is a point of attraction of I.

Example 3.3
Determine whether $x = 1$ and $x = 4$ are points of attraction of the iterative function of equation (3.8).

Solution 3.3 The derivative of the iterative process I in equation (3.8) is

$$\frac{\partial g(x)}{\partial x} = \frac{2}{5}x$$

Thus, for $x = 1$, $\frac{2}{5}x = \frac{2}{5} < 1$ and $x = 1$ is a point of attraction of I. For $x = 4$: $\frac{2}{5}x = \frac{2}{5}(4) = \frac{8}{5} > 1$; thus, $x = 4$ is not a point of attraction of I.

There are four possible convergence types for fixed point iterations. These are shown graphically in Figure 3.4. Figure 3.4 (a) shows what happens if $g'(x)$ is between 0 and 1. Even if the initial guess x_0 is far from x, the successive values of x^k approach the solution from one side … this is defined as monotonic convergence Figure 3.4 (b) shows the situation when $g'(x)$ is

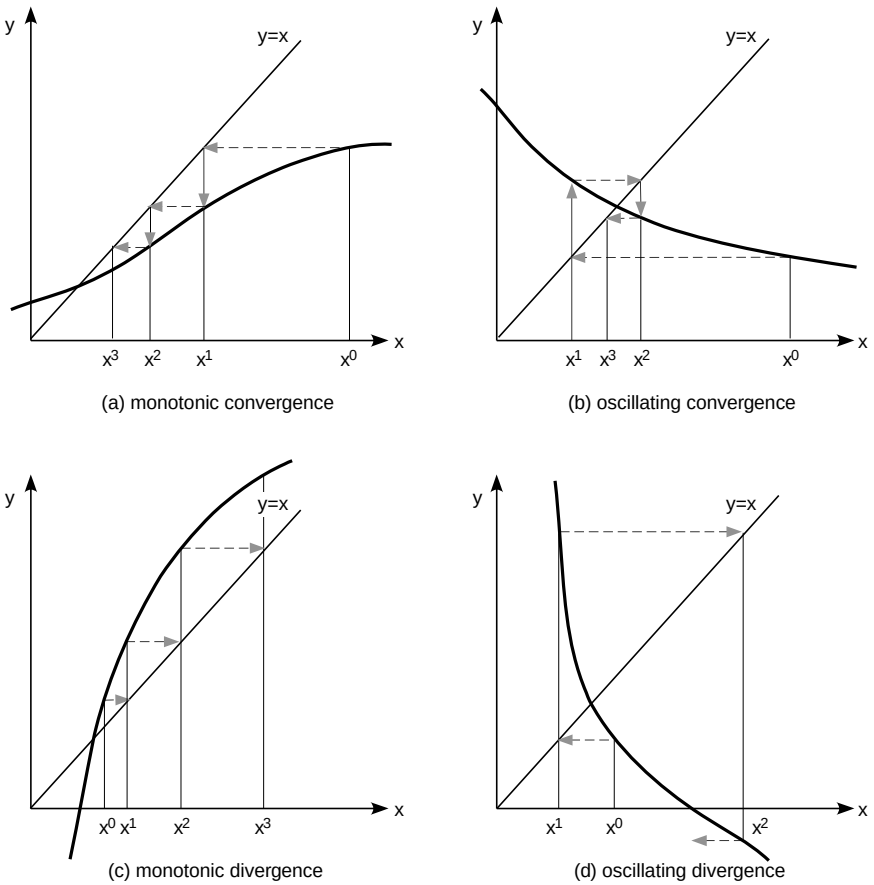


FIGURE 3.4
Four possible convergence types in the iteration $x = g(x)$

between -1 and 0. Even if the initial guess x_0 is far from x, the successive values of x^k approach the solution from "rst one side and then the other oscillating around the root. This convergence is *oscillatory convergence*.

Figure 3.4 (c) shows the case when $g(x)$ is greater than 1 leading to monotonic divergence. Figure 3.4 (d) illustrates the case when $g(x) < -1$ and $|g(x)| > 1$. This is oscillatory divergence.

3.2 Newton-Raphson Iteration

Several iterative methods offer more robust convergence behavior than the simple fixed point iteration described in the previous section. One of the most widely used iterative methods is the Newton-Raphson iterative method. This method can also be described by the iterative process

$$I: \quad x^{k+1} = g\left(x^k\right), \quad k = 1, \ldots,$$

but frequently offers better convergence properties than the fixed point iteration.

Consider again the scalar nonlinear function

$$f(x) = 0 \tag{3.20}$$

Expanding this function in a Taylor's series expansion about the point x^k yields

$$f(x) = f\left(x^k\right) + \left.\frac{\partial f}{\partial x}\right|_{x^k}\left(x - x^k\right) + \frac{1}{2!}\left.\frac{\partial^2 f}{\partial x^2}\right|_{x^k}\left(x - x^k\right)^2 + \ldots = 0 \tag{3.21}$$

If it is assumed that the iterates will converge to x as $k \to \infty$, then the updated guess x^{k+1} can be substituted for x, yielding

$$f\left(x^{k+1}\right) = f\left(x^k\right) + \left.\frac{\partial f}{\partial x}\right|_{x^k}\left(x^{k+1} - x^k\right) + \frac{1}{2!}\left.\frac{\partial^2 f}{\partial x^2}\right|_{x^k}\left(x^{k+1} - x^k\right)^2 + \ldots = 0 \tag{3.22}$$

If the initial guess is sufficiently close to x and within the domain of attraction of x, then the higher order terms of the expansion can be neglected, yielding

$$f\left(x^{k+1}\right) = f\left(x^k\right) + \left.\frac{\partial f}{\partial x}\right|_{x^k}\left(x^{k+1} - x^k\right) \approx 0 \tag{3.23}$$

Solving directly for x^{k+1} as a function of x^k yields the following iterative function:

$$I: \quad x^{k+1} = x^k - \left.\frac{\partial f}{\partial x}\right|_{x^k}^{-1} f\left(x^k\right) \tag{3.24}$$

which is the well-known Newton-Raphson iterative method.

The Newton-Raphson method also lends itself to a graphical interpretation. Consider the same function as in Example 3.2 plotted in Figure 3.5. In this method, the slope of the function evaluated at the current iteration is used to produce the next guess. For any guess x^k, there corresponds a point on the function $f\left(x^k\right)$ with slope

$$\left.\frac{\partial f}{\partial x}\right|_{x = x^k}$$

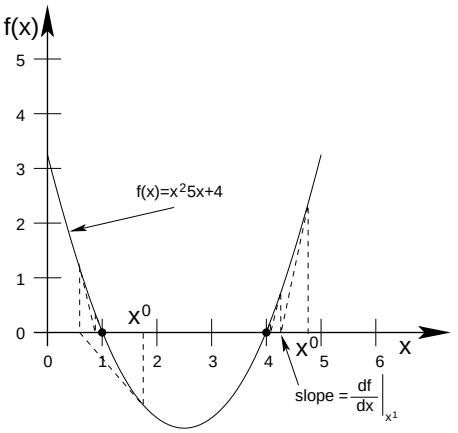


FIGURE 3.5
Graphical interpretation of the Newton-Raphson method

Therefore, the next guess x^{k+1} is simply the intersection of the slope and the x-axis. This process is repeated until the guesses are sufficiently close to the solution x . An iteration is said to have converged at x^k if

$$f\ x^k\ <$$

where is some pre-determined tolerance.

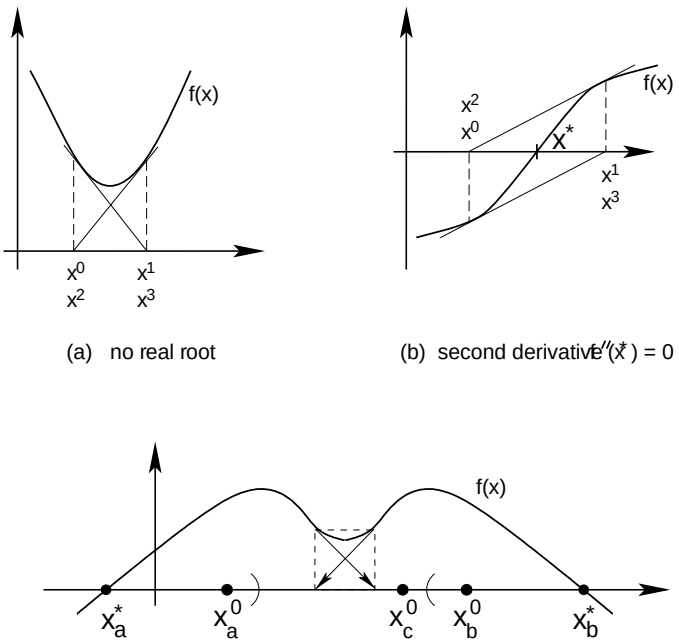
Example 3.4
Repeat Example 3.2 using a Newton-Raphson iteration.

Solution 3.4 Using the Newton-Raphson method of equation (3.24), the iterative function is given by:

$$I: \quad x^{k+1} = x^k - \frac{x^{k\ 2} - 5x^k + 4}{2x^k - 5} \tag{3.25}$$

Using this iterative function, the estimates to x from an initial guess of $x^0 = 3$ are:

k	x^k	g x^k
0	3	$3 - \frac{9-15+4}{6-5} = 5$
1	5	$5 - \frac{25-25+4}{10-5} = 4.2$
2	4.2	$4.2 - \frac{17.64-21+4}{8.4-5} = 4.012$



(a) no real root

(b) second derivative $f''(x^*) = 0$

(c) x^0 not close enough to root

FIGURE 3.6
Newton-Raphson regions of convergence

Similarly, the estimates to x from an initial guess of $x^0 = 2$ are:

k	x^k	$g(x^k)$
0	2	$2 - \frac{4^3 - 10 \cdot 4 + 4}{4^2 - 5} = 0$
1	0	$0 - \frac{0^3 - 0 + 4}{0^2 - 5} = 0.8$
2	0.8	$0.8 - \frac{0.64^3 - 4 + 4}{1.6^2 - 5} = 0.988$

In this case, both solutions are points of attraction of the Newton-Raphson iteration.

In some cases however, the Newton-Raphson method will also fail to converge. Consider the functions shown in Figure 3.6. In Figure 3.6 (a), the function has no real root. In Figure 3.6(b), the function is symmetric around x and the second derivative is zero. In Figure 3.6 (c), an initial guess of x_a^0 will converge to the solution x_a. An initial guess of x_b^0 will converge to

the solution x_b. However, an initial guess of x_c^0 will cause the iterates to get locked-in and oscillate in the region denoted by the dashed box without ever converging to a solution. This figure supports the assertion that if the initial guess is too far away from the actual solution, the iterates may not converge. Or conversely, the initial guess must be sufficiently close to the actual solution for the Newton-Raphson iteration to converge. This supports the initial assumption used to derive the Newton-Raphson algorithm in that if the iterates were sufficiently close to the actual solution, the higher-order terms of the Taylor series expansion could be neglected. If the iterates are not sufficiently close to the actual solution, these higher-order terms are significant and the assumption upon which the Newton-Raphson algorithm is based is not valid.

3.2.1 Convergence Properties

Note that the rate of convergence to the solution in Example 3.4 is much faster than in Example 3.2. This is because the Newton-Raphson method exhibits quadratic convergence, whereas the fixed-point iteration exhibits only linear convergence. Linear convergence implies that once the iterates x^k are sufficiently close to the actual solution x^*, then the error

$$\varepsilon^k = x^k - x^* \tag{3.26}$$

will approach zero in a linear fashion. The convergence of Examples 3.2 and 3.4 is shown in Figure 3.7. Plotted on a log-scale plot, the error for the fixed point iteration is clearly linear, whereas the Newton-Raphson error exhibits quadratic convergence until it becomes too small to plot. Numerous methods have been proposed to predict the rate of convergence of iterative methods. Let the error of an iterative function be defined as in equation (3.26). If there exists a number p and a constant $C \neq 0$ such that

$$\lim_k \frac{|\varepsilon^{k+1}|}{|\varepsilon^k|^p} = C \tag{3.27}$$

then p is called the order of convergence of the iterative sequence and C is the asympototic error constant. If $p = 1$, the convergence is said to be linear. If $p = 2$, the convergence is quadratic, and if $p = 3$, the order of convergence is cubic. The Newton-Raphson method satisfies equation (3.27) with $p = 2$ if

$$C = \frac{1}{2}\frac{\frac{d^2 f(x^*)}{dx^2}}{\frac{df(x^*)}{dx}}$$

where $C \neq 0$ only if $\frac{d^2 f(x^*)}{dx^2} \neq 0$. Thus, for most functions, the Newton-Raphson method exhibits quadratic convergence.

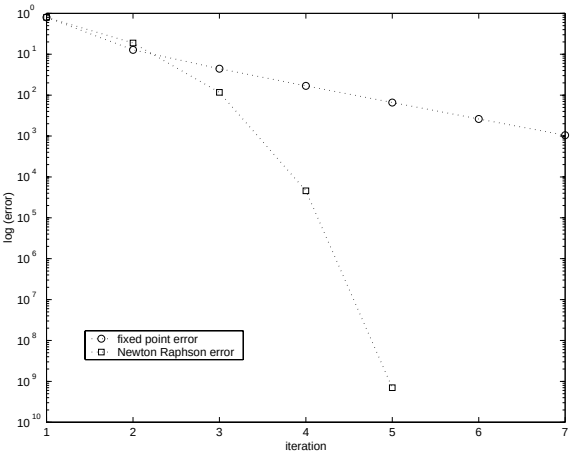


FIGURE 3.7
Non-converging iteration (“xed point vs. Newton-Raphson)

3.2.2 The Newton-Raphson for Systems of Nonlinear Equations

In science and engineering, many applications give rise to systems of equations such as those in equation (3.2). With a few modi“cations, the Newton-Raphson method developed in the previous section can be extended to systems of nonlinear equations. Systems of equations can similarly be represented by Taylor series expansions. By making the assumption once again that the initial guess is su ciently close to the exact solution, then the multi-dimensional higher order terms can be neglected, yielding the Newton-Raphson method for n-dimensional systems:

$$x^{k+1} = x^k - J\left(x^k\right)^{-1} F\left(x^k\right) \tag{3.28}$$

where

$$x = \begin{matrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{matrix}$$

$$F\left(x^k\right) = \begin{matrix} f_1\left(x^k\right) \\ f_2\left(x^k\right) \\ f_3\left(x^k\right) \\ \vdots \\ f_n\left(x^k\right) \end{matrix}$$

and the Jacobian matrix $J\ x^k$ is given by

$$J\ x^k = \begin{matrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} & \cdots & \frac{\partial f_3}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_n}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_n} \end{matrix}$$

Typically the inverse of the Jacobian $J\ x^k$ is not found directly, but rather through LU factorization by posing the Newton-Raphson method as

$$J\ x^k \quad x^{k+1} - x^k = -F\ x^k \tag{3.29}$$

which is now in the form $Ax = b$ where the Jacobian is the matrix A, the function $-F\ x^k$ is the vector b, and the unknown x is the difference vector $x^{k+1} - x^k$. Convergence is typically evaluated by considering the norm of the function

$$F\ x^k < \tag{3.30}$$

Note that the Jacobian is a function of x^k and is therefore updated every iteration along with $F\ x^k$.

Example 3.5
Find the solution to

$$0 = x_1^2 + x_2^2 - 5x_1 + 1 = f_1(x_1, x_2) \tag{3.31}$$
$$0 = x_1^2 - x_2^2 - 3x_2 - 3 = f_2(x_1, x_2) \tag{3.32}$$

with an initial guess of

$$x^{(0)} = \begin{matrix} 3 \\ 3 \end{matrix}$$

Solution 3.5 The Jacobian of this system of equations is:

$$J(x_1, x_2) = \begin{matrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{matrix} = \begin{matrix} 2x_1 - 5 & 2x_2 \\ 2x_1 & -2x_2 - 3 \end{matrix}$$

Iteration 1

The Jacobian and the functions f_1 and f_2 are evaluated at the initial condition

$$\begin{matrix} 1 & 6 \\ 6 & -9 \end{matrix} \begin{matrix} x_1^{(1)} - 3 \\ x_2^{(1)} - 3 \end{matrix} = \begin{matrix} -4 \\ 12 \end{matrix} \tag{3.33}$$

Solving this linear system yields

$$\begin{matrix} x_1^{(1)} - 3 \\ x_2^{(1)} - 3 \end{matrix} = \begin{matrix} 0.8 \\ -0.8 \end{matrix} \tag{3.34}$$

Thus

$$x_1^{(1)} = 0.8 + x_1^{(0)} = 0.8 + 3 = 3.8 \tag{3.35}$$

$$x_2^{(1)} = -0.8 + x_2^{(0)} = -0.8 + 3 = 2.2 \tag{3.36}$$

The error at iteration 1 is

$$\begin{matrix} f_1\left(x_1^{(0)}, x_2^{(0)}\right) \\ f_2\left(x_1^{(0)}, x_2^{(0)}\right) \end{matrix} = 12$$

Iteration 2

The Jacobian and the functions f_1 and f_2 are evaluated at $x^{(1)}$

$$\begin{matrix} 2.6 & 4.4 \\ 7.6 & -7.4 \end{matrix} \begin{matrix} x_1^{(2)} - 3.8 \\ x_2^{(2)} - 2.2 \end{matrix} = \begin{matrix} -1.28 \\ 0.00 \end{matrix} \tag{3.37}$$

Solving this linear system yields

$$\begin{matrix} x_1^{(2)} - 3.8 \\ x_2^{(2)} - 2.2 \end{matrix} = \begin{matrix} -0.1798 \\ -0.1847 \end{matrix} \tag{3.38}$$

Thus

$$x_1^{(2)} = -0.1798 + x_1^{(1)} = -0.1798 + 3.8 = 3.6202 \tag{3.39}$$

$$x_2^{(2)} = -0.1847 + x_2^{(1)} = -0.1847 + 2.2 = 2.0153 \tag{3.40}$$

The error at iteration 2 is

$$\begin{matrix} f_1\left(x_1^{(1)}, x_2^{(1)}\right) \\ f_2\left(x_1^{(1)}, x_2^{(1)}\right) \end{matrix} = 1.28$$

Iteration 3

The Jacobian and the functions f_1 and f_2 are evaluated at $x^{(2)}$

$$\begin{matrix} 2.2404 & 4.0307 \\ 7.2404 & -7.0307 \end{matrix} \begin{matrix} x_1^{(3)} - 3.6202 \\ x_2^{(3)} - 2.0153 \end{matrix} = \begin{matrix} -0.0664 \\ 0.0018 \end{matrix} \tag{3.41}$$

Solving this linear system yields

$$\begin{matrix} x_1^{(3)} - 3.6202 \\ x_2^{(3)} - 2.0153 \end{matrix} = \begin{matrix} -0.0102 \\ -0.0108 \end{matrix} \tag{3.42}$$

Thus

$$x_1^{(3)} = \check{S}0.0102 + x_1^{(2)} = \check{S}0.0102 + 3.6202 = 3.6100 \quad (3.43)$$

$$x_2^{(3)} = \check{S}0.0108 + x_2^{(2)} = \check{S}0.0108 + 2.0153 = 2.0045 \quad (3.44)$$

The error at iteration 3 is

$$\begin{matrix} f_1 & x_1^{(2)}, x_2^{(2)} \\ f_2 & x_1^{(2)}, x_2^{(2)} \end{matrix} = 0.0664$$

At iteration 4, the functions f_1 and f_2 are evaluated at $x^{(3)}$ and yield the following:

$$\begin{matrix} f_1 & x_1^{(3)}, x_2^{(3)} \\ f_2 & x_1^{(3)}, x_2^{(3)} \end{matrix} = \begin{matrix} \check{S}0.221 \times 10^{\check{S}3} \\ 0.012 \times 10^{\check{S}3} \end{matrix}$$

Since the norm of this matrix is very small, it can be concluded that the iterates have converged and

$$\begin{matrix} x_1^{(3)} \\ x_2^{(3)} \end{matrix} = \begin{matrix} 3.6100 \\ 2.0045 \end{matrix}$$

are within an order of error of $10^{\check{S}3}$ of the actual solution.

In the solution to Example 3.5, the error at each iteration is

iteration	error
0	12.0000
1	1.2800
2	0.0664
3	0.0002

Note that once the solution is su ciently close to the actual solution, the error at each iteration decreases rapidly. f the iterations were carried far enough the error at each iteration would become roughly the square of the previous iteration error. This convergence behavior is indicative of the quadratic convergence of the Newton-Raphson method.

3.2.3 Modi“cations to the Newton-Raphson Method

Although the full Newton-Raphson method exhibits quadratic convergence and a minimum number of iterations, each iteration may require signi“cant computation. For example, the computation of a full Jacobian matrix requires n^2 calculations, and each iteration requires on the order of n^3 operations for the LU factorization if the Jacobian is a full matrix. Therefore most modi“cations to the Newton-Raphson method propose to reduce either the calculation or the LU factorization of the Jacobian matrix.

Consider once again the iterative statement of the Newton-Raphson method:

$$I: \quad x^{k+1} = x^k - J\left(x^k\right)^{-1} f\left(x^k\right)$$

This iterative statement can be written in a more general form as:

$$I: \quad x^{k+1} = x^k - M\left(x^k\right)^{-1} f\left(x^k\right) \tag{3.45}$$

where M is an $n \times n$ matrix that may or may not be a function of x^k. Note that even if $M = J$, this iteration will still converge to a correct solution for x if the function $f(x)$ is driven to zero. So one approach to simplifying the Newton-Raphson method is to find a suitable substitute matrix M that is easier to compute than the system Jacobian. One common simplification is to substitute each of the partial derivative entries $\frac{f_i}{x_j}$ by a difference approximation. For example, a simple approximation might be

$$\frac{f_i}{x_j} \approx \frac{1}{h_{ij}} f_i\left(x + h_{ij} e^j\right) - f_i(x) \tag{3.46}$$

where e^j is the j^{th} unit vector:

$$e^j = \begin{matrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{matrix}$$

where the 1 occurs in the j^{th} row of the unit vector and all other entries are zero. The scalar h_{ij} can be chosen in numerous ways, but one common choice is to let $h_{ij}^k = x_j^k - x_j^{k-1}$. This choice for h_{ij} leads to a rate of convergence of 1.62, which lies between quadratic and linear convergence rates.

Another common modification to the Newton-Raphson method is to set M equal to the Jacobian matrix at occasional intervals. For example, the matrix M can be re-evaluated whenever the convergence slows down, or at more regular intervals, such as every other or other third iteration. This modification is known as the dishonest Newton's method. An extreme extension of this method is to set M equal to the initial Jacobian matrix and then to hold it constant throughout the remainder of the iteration. This is commonly called the very dishonest Newton's method. In addition to the reduction in computation associated with the calculation of the matrix, this method also has the advantage that the M matrix need only be factored into the LU matrices once since it is a constant. This can save considerable computation time in the LU factorization process. Similarly, the matrices of the dishonest Newton's method need only be factored when the M matrix is re-evaluated.

3.3 Continuation Methods

Many of the iteration methods described so far will, in general, converge to a solution x^* of $f(x) = 0$ only if the initial condition is sufficiently close to x^*. The continuation method approach may be considered to be an attempt to widen the region of convergence of a given method. In many physical systems, the problem defined by the mathematical equation $f(x) = 0$ may in some way depend in a natural way on a parameter λ of the system. When this parameter is set equal to 0, the system $f_0(x) = 0$ has a known solution x^0. However, for varying λ, and entire family of functions $H(x, \lambda)$ exist such that:

$$H(x, 0) = f_0(x), \quad H(x, 1) = f(x) \tag{3.47}$$

where a solution x^0 of $H(x, 0) = 0$ is known, and the equation $H(x, 1) = 0$ is the desired problem to be solved.

Even if $f(x)$ does not depend naturally on a suitable parameter λ, a family of problems satisfying equation (3.47) can be defined by

$$H(x, \lambda) = \lambda f(x) + (1 - \lambda) f_0(x), \qquad \lambda \in [0, 1] \tag{3.48}$$

when the solution x^0 of $f_0(x) = 0$ is known. As λ varies from 0 to 1, the family of mappings varies from $f_0(x) = 0$ to $f_1(x) = 0$ where the solution of $f_1(x) = f(x) = 0$ is the desired value $x^1 = x^*$.

As a first approach to obtaining $x^* = x^1$, the interval $[0, 1]$ can be partitioned as

$$0 = \lambda_0 < \lambda_1 < \lambda_2 < \ldots < \lambda_N = 1$$

and consider solving the problems

$$H(x, \lambda_i) = 0, \quad i = 1, \ldots, N \tag{3.49}$$

and assuming that a Newton-Raphson iteration is used to solve each problem i in equation (3.49), then the initial condition for the i-th problem is the solution from $H(x, \lambda_{i-1}) = 0$. For small enough intervals between i and $i + 1$, this solves the problem of identifying a good initial condition.

The relationship given in equation (3.48) is an example of a *homotopy* in which two functions $f(x)$ and $f_0(x)$ are embedded in a single continuous function. Formally, a homotopy between any two functions is a continuous mapping $f, f_0 : X \to Y$:

$$H : [0, 1] \times X \to Y \tag{3.50}$$

such that equation (3.47) holds. If such a mapping exists, then it is said that f is homotopic to f_0.

Homotopy functions are used to define a path from the known solution to a relatively simple problem ($f_0(x) = 0$) to the solution of a more complex

problem to which the solution is desired ($f(x) = 0$). This path comprises the solutions to a family of problems which represent the continuous deformation from the simple problem to the desired problem. Continuation methods are a numerical approach to following the deformation path.

Homotopy continuation methods can be constructed to be exhaustive and globally convergent, meaning that all solutions to a given system of nonlinear equations can be found and will converge regardless of choice of initial condition [58]. Since a homotopy problem is equal to zero at every point $[0, 1]$ along the path, it is therefore equal to zero at $= _k$ and $= _{k+1}$ which are two successive points along the path. This gives:

$$0 = H\ x^k, _k = _k f(x^k) + (1 \check{S}\ _k) f_0(x^k) \quad (3.51)$$

$$0 = H\ x^{k+1}, _{k+1} = _{k+1} f(x^{k+1}) + (1 \check{S}\ _{k+1}) f_0(x^{k+1}) \quad (3.52)$$

For paths along the path, the homotopy parameter $_{k+1}$ is associated with the parameter set

$$x^{k+1} = x^k + \ x$$

If the changes in the parameters are small, the functions $f_0(x^{k+1})$ and $f(x^{k+1})$ can be linearly approximated by using a Taylor series expansion about x^k and neglecting all terms higher than second order. Applying this technique to equation (3.52) yields:

$$(\ _k + \ ) F_x\ x^k \ \ x + (1 \check{S}\ _k \check{S}\ ) f_0\ x^k + F_{0x}\ x^k \ \ x = 0 \quad (3.53)$$

where F_x and F_{0x} are the Jacobians of $f(x)$ and $f_0(x)$ with respect to x, respectively. Subtracting equation (3.51) from equation (3.53) and canceling like terms yields

$$0 = \ _{k+1} F_x\ x^k + (1 \check{S}\ _{k+1}) F_{0x}\ x^k \ \ x + f\ x^k \check{S} f_0\ x^k \quad (3.54)$$

Using $x^{k+1} = x^k + \ x$, equation (3.54) can be rewritten in terms of the homotopy function to yield the update equation:

$$x^{k+1} = x^k \check{S}\ H_x\ x^k, _{k+1}^{\check{S}1}\ H\ x^k, _{k+1} \quad (3.55)$$

where

$$_{k+1} = _k +$$

and $H_x(x,)$ is the Jacobian of $H(x,)$ with respect to x.

Example 3.6
Solve

$$0 = f_1(x_1, x_2) = x_1^2 \check{S} 3x_2^2 + 3 \quad (3.56)$$

$$0 = f_2(x_1, x_2) = x_1 x_2 + 6 \quad (3.57)$$

using the homotopic mapping with

$$0 = f_{01}(x_1, x_2) = x_1^2 - 4 \tag{3.58}$$
$$0 = f_{02}(x_1, x_2) = x_2^2 - 9 \tag{3.59}$$

Solution 3.6 Set up a homotopy such that

$$H(x, \lambda) = \lambda f(x) + (1 - \lambda) f_0(x), \qquad \lambda \in [0, 1] \tag{3.60}$$

$$0 = \lambda (x_1^2 - 3x_2^2 + 3) + (1 - \lambda)(x_1^2 - 4) \tag{3.61}$$
$$0 = \lambda (x_1 x_2 + 6) + (1 - \lambda)(x_2^2 - 9) \tag{3.62}$$

The continuation method advances the solution via equation (3.55):

$$\lambda^{k+1} = \lambda^k + \Delta\lambda \tag{3.63}$$

$$\begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \end{bmatrix} = \begin{bmatrix} x_1^k \\ x_2^k \end{bmatrix} - \lambda^{k+1} \begin{bmatrix} 2x_1^k & -6x_2^k \\ x_2^k & x_1^k \end{bmatrix} + (1 - \lambda^{k+1}) \begin{bmatrix} 2x_1^k & 0 \\ 0 & 2x_2^k \end{bmatrix}^{-1} \times \begin{bmatrix} (x_1^k)^2 - 3(x_2^k)^2 + 3 - \lambda (x_1^k)^2 - 4 \\ x_1^k x_2^k + 6 - \lambda (x_2^k)^2 - 9 \end{bmatrix} \tag{3.64}$$

The solution is then re“ned through the Newton-Raphson solution for x_1^{k+1} and x_2^{k+1} from

$$0 = \lambda^{k+1} (x_1^{k+1})^2 - 3(x_2^{k+1})^2 + 3 + (1 - \lambda^{k+1}) (x_1^{k+1})^2 - 4 \tag{3.65}$$
$$0 = \lambda^{k+1} x_1^{k+1} x_2^{k+1} + 6 + (1 - \lambda^{k+1}) (x_2^{k+1})^2 - 9 \tag{3.66}$$

Starting with $\lambda^0 = 0$ and $\Delta\lambda = 0.1$ yields the easily obtained initial solution of the system:

$$x_1^0 = 2$$
$$x_2^0 = 3$$

Predicting the values for k = 1 from equations (3.63) and (3.64) yields:

$$x_1^1 = 2.3941$$
$$x_2^1 = 2.7646$$

Re“ning the solution via the Newton-Raphson solution to equations (3.65)-(3.66) yields

$$x_1^1 = 2.3628$$
$$x_2^1 = 2.7585$$

This process is repeated until = 1 and $x_1 = -3$ and $x_2 = 2$ which are the correct solutions to the desired problem.

The same process will work if the inital solutions are chosen as $x_1^0 = -2$ and $x_2^0 = -3$. In this case, the obtained values are the alternate solution $x_1 = 3$ and $x_2 = -2$ to the desired problem.

3.4 Secant Method

The Newton-Rapshon method is based on using the line tangent to the function $y = f(x)$. By using the slope of the tangent line, the update to the iteration can be calculated. The difficulty with this method is the computational complexity required to compute the function derivative, or $f'(x)$. An alternate approach to calculating the slope of the tangent is to take two points close to the desired root and interpolate between them to estimate the slope as shown in Figure 3.8. This produces the linear function

$$q(x) = a_0 + a_1 x \tag{3.67}$$

where $q(x^0) = f(x^0)$ and $q(x^1) = f(x^1)$. This line is the secant line and is given by

$$q(x) \frac{x^1 - x}{x^1 - x^0} f(x^0) + \frac{x - x^0}{x^1 - x^0} f(x^1) \tag{3.68}$$

Setting $x_2 = x$ and solving yields:

$$x^2 = x^1 - f(x^1) \left[\frac{f(x^1) - f(x^0)}{x^1 - x^0} \right]^{-1} \tag{3.69}$$

The process can now be repeated by using x^2 and x^1 to produce another secant line. By repeatedly updating the secant line, the generalized formula becomes:

$$x^{k+1} = x^k - f(x^k) \left[\frac{f(x^k) - f(x^{k-1})}{x^k - x^{k-1}} \right]^{-1} \tag{3.70}$$

Note that the secant method can be considered an approximation of the Newton-Raphson method

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)} \tag{3.71}$$

by using the approximation

$$f'(x^k) = \frac{f(x^k) - f(x^{k-1})}{x^k - x^{k-1}} \tag{3.72}$$

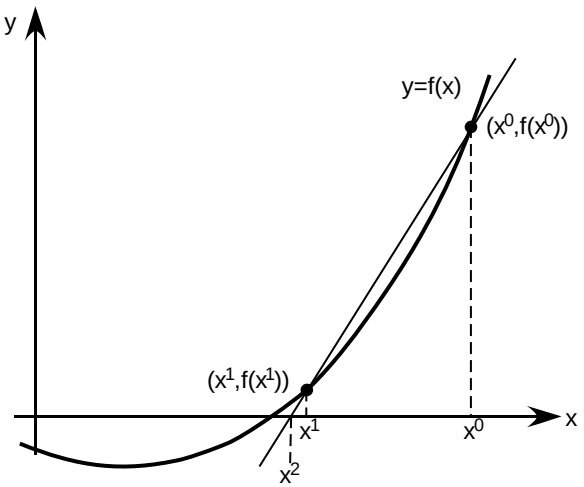


FIGURE 3.8
Illustration of secant method

The secant method is often faster than the Newton-Raphson method even though it requires a greater number of iterations to converge to the same level of accuracy. This is because the Newton-Raphson method requires two function evaluations $f\ x^k$ and $f'\ x^k$ whereas the secant method only requires one function evaluation $f\ x^k$ since $f\ x^{k-1}$ can be saved from the previous iteration.

The secant method exhibits super linear convergence; its convergence is faster than linear convergence, but not as fast as quadratic convergence (of the Newton-Raphson method). Let the error at iteration k be given by

$$e^k = x^k - x \tag{3.73}$$

where x is the exact solution. Using the Taylor series expansion:

$$f\ x^k = f\ x + x^k - x \tag{3.74}$$

$$= f\ x + e^k \tag{3.75}$$

$$= f\ (x\ ) + f'\ (x\ )\, e^k + \frac{1}{2} f''\ (x\ )\ e^k{}^2 + \ldots \tag{3.76}$$

Similarly

$$f\ x^{k-1} = f\ (x\ ) + f'\ (x\ )\, e^{k-1} + \frac{1}{2} f''\ (x\ )\ e^{k-1}{}^2 + \ldots \tag{3.77}$$

Furthermore

$$x^k - x^{k-1} = x^k - x - x^{k-1} - x = e^k - e^{k-1}$$

Subtracting x from both sides of equation (3.70) and recalling that $f(x) = 0$ yields

$$e^{k+1} = e^k - \frac{f(x)e^k + \frac{1}{2}f''(x)(e^k)^2}{f'(x) + \frac{1}{2}f''(x)(e^k + e^{k-1})} \tag{3.78}$$

or

$$e^{k+1} = \frac{1}{2}\frac{f''(x)}{f'(x)}e^k e^{k-1} + O(e^3) \tag{3.79}$$

Let

$$e^k = C_k (e^k)^r$$

where r is the convergence order. If $r > 1$, then the convergence rate is super linear. If the remainder term in equation (3.79) is negligible, then equation (3.79) can be rewritten

$$\frac{e^{k+1}}{e^k e^{k-1}} = \frac{1}{2}\frac{f''(x)}{f'(x)} \tag{3.80}$$

and in the limit

$$\lim_k \frac{e^{k+1}}{e^k e^{k-1}} = C \tag{3.81}$$

For large k,

$$e^k = C (e^{k-1})^r$$

and

$$e^{k+1} = C (e^k)^r = C C (e^{k-1})^{r r} = C^{r+1} (e^{k-1})^{r^2}$$

Substituting this into equation (3.81) yields

$$\lim_k C^r (e^{k+1})^{r^2 - r - 1} = C \tag{3.82}$$

Since $\lim_k e^k = 0$, this relationship can only be satisfied if $r^2 - r - 1 = 0$, which has the solution

$$r = \frac{1 + \sqrt{5}}{2} > 1 \tag{3.83}$$

and hence superlinear convergence.

Example 3.7
Use the secant method to find a solution of

$$0 = e^{x^2 - 2} - 3\ln(x)$$

starting with $x^0 = 1.5$ and $x^1 = 1.4$.

Solution 3.7 Using equation (3.70), the following set of values are obtained.

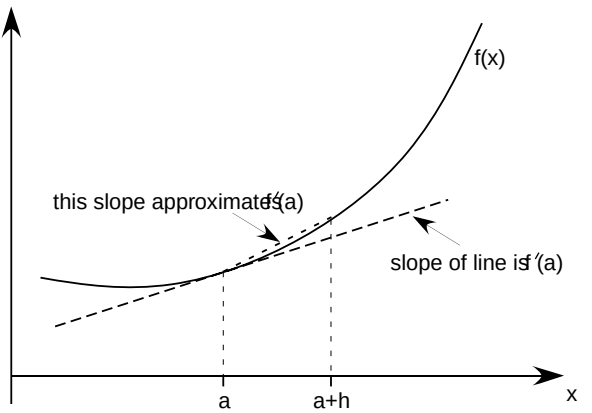


FIGURE 3.9
Graphical interpretation of the di erence approximation of the slope of f (a)

k	x^{k+1}	x^{k}	x^{k-1}	$f(x^{k})$	$f(x^{k+1})$
1	1.4418	1.4000	1.5000	-0.0486	0.0676
2	1.4617	1.4418	1.4000	-0.0157	-0.0486
3	1.4552	1.4617	1.4418	0.0076	-0.0157
4	1.4557	1.4552	1.4617	-0.0006	0.0076
5	1.4557	1.4557	1.4552	-0.0000	-0.0006

3.5 Numerical Di erentiation

Using the Newton-Raphson method or any of its modi“cations requires the calculation of numerous partial derivatives. In many cases, the analytic computation of the partial derivative may be extremely complex or may be computationally expensive to compute. In these cases, it is desirable to compute the derivative numerically directly from the function f (x) without explicit knowledge of $\frac{f}{x}$.

Consider the scalar function f (x). The derivative of the function f at the point x = a is equivalent to the slope of the function f (a). A reasonable approximation to the slope of a curve f (a) is to use a nearby point a + h to compute a di erence approximation as shown in Figure 3.9.

This has a mathematical basis that can be derived by application of the Taylor series expansion tof (a + h):

$$f(a+h) = f(a) + h\frac{f}{x}(a) + \frac{h^2}{2!}\frac{^2f}{x^2}(a) + \frac{h^3}{3!}\frac{^3f}{x^3}(a) + \ldots \tag{3.84}$$

By rearranging:

$$\frac{f(a+h) - f(a)}{h} = \frac{f}{x}(a) + \frac{h}{2!}\frac{^2f}{x^2}(a) + \ldots \tag{3.85}$$

By neglecting the higher order terms:

$$\frac{f}{x}(a) \quad \frac{f(a+h) - f(a)}{h} \tag{3.86}$$

This approximation becomes increasingly more accurate ash becomes smaller (and is exact in the limit as h 0). This approximation is the one-sided di erence approximation known as the forward di erence approximation to the derivative of f . A similar approach can be taken in which the series is expanded about a h and

$$\frac{f}{x}(a) \quad \frac{f(a) - f(a-h)}{h} \tag{3.87}$$

which is the approximation known as the backward di erence approximation. Consider now the combination of the two approaches:

$$\frac{f}{x}(a) \quad \frac{f(a+h) - f(a-h)}{2h} \tag{3.88}$$

This combination is often referred to as thecenter di erence approximation. The forward and backward di erence approximations both have error on the order of)(h), whereas the center approximation has an error on the order of $O\ h^2$ and will in general have better accuracy than either the forward or backward di erence approximations.

Example 3.8
Consider the polynomial

$$f(x) = x^3 + x^2 - \frac{5}{4}x - \frac{3}{4}$$

Approximate the derivative of this polynomial in the range [2, 1.5] with h = 0 .2 using the forward, backward, and center di erence approximations.

Solution 3.4 The exact derivative of this function is given by

$$f\ (x) = 3x^2 + 2x - \frac{5}{4}$$

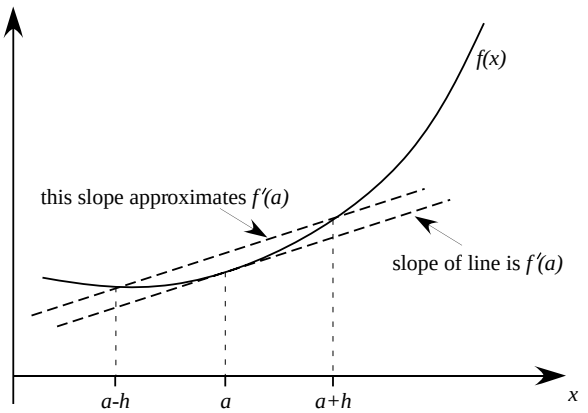


FIGURE 3.10
Graphical interpretation of the center di erence approximation of the slope of f (a)

x	f (x Š h)	f (x)	f (x + h)	f (x) backward	f (x) forward	f (x) center	f (x) exact
-2.0	-3.808	-2.250	-1.092	7.79	5.79	6.79	6.75
-1.8	-2.250	-1.092	-0.286	5.79	4.03	4.91	4.87
-1.6	-1.092	-0.286	0.216	4.03	2.51	3.27	3.23
-1.4	-0.286	0.216	0.462	2.51	1.23	1.87	1.83
-1.2	0.216	0.462	0.500	1.23	0.19	0.71	0.67
-1.0	0.462	0.500	0.378	0.19	-0.61	-0.21	-0.25
-0.8	0.500	0.378	0.144	-0.61	-1.17	-0.89	-0.93
-0.6	0.378	0.144	-0.154	-1.17	-1.49	-1.33	-1.37
-0.4	0.144	-0.154	-0.468	-1.49	-1.57	-1.53	-1.57
-0.2	-0.154	-0.468	-0.750	-1.57	-1.41	-1.49	-1.53
-0.0	-0.468	-0.750	-0.952	-1.41	-1.01	-1.21	-1.25
0.2	-0.750	-0.952	-1.026	-1.01	-0.37	-0.69	-0.73
0.4	-0.952	-1.026	-0.924	-0.37	0.51	0.07	0.03
0.6	-1.026	-0.924	-0.598	0.51	1.63	1.07	1.03
0.8	-0.924	-0.598	-0.000	1.63	2.99	2.31	2.27
1.0	-0.598	-0.000	0.918	2.99	4.59	3.79	3.75
1.2	-0.000	0.918	2.204	4.59	6.43	5.51	5.47
1.4	0.918	2.204	3.906	6.43	8.51	7.47	7.43

Figure 3.11 clearly shows the accuracy levels of the di erent derivative approximations.

By continuing in the same approach of using Taylor series expansions and including additional information, increasingly more accurate approximations can be achieved. One such approximation that is widely used is the *Richardson*

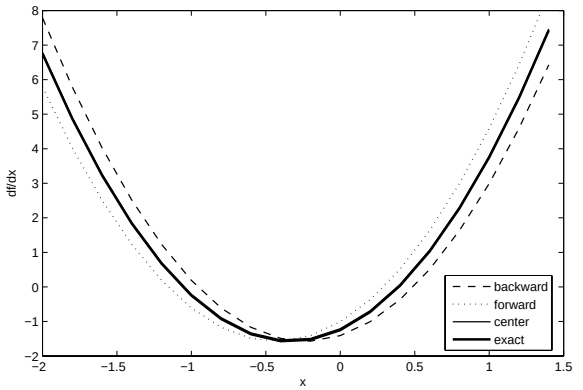


FIGURE 3.11
Exact versus approximate derivatives

approximation:

$$f'(x) \approx \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h} \tag{3.89}$$

This approximation has an error on the order $O\left(h^4\right)$.

Consider once again the Newton-Raphson method which requires the calculation of the Jacobian. The approximations can be utilized to calculate the derivatives in the Jacobian rather than a direct analytic calculation. For example, consider the system of nonlinear equations:

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

The Jacobian for this system consists of partial derivatives of the form $\frac{\partial f_i}{\partial x_j}$ which can now be approximated using one of the approximation methods introduced. For example, using the center di erence:

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(x_j + \Delta x_j) - f_i(x_j - \Delta x_j)}{2\Delta x_j}$$

where Δx_j is usually chosen to be a small incremental change (of about 1%).

3.6 Power System Applications

The solution and analysis procedures outlined in this chapter form the basis of a set of powerful tools that can be used for a myriad of power system applications. One of the most outstanding features of power systems is that they are modeled as an extremely large set of nonlinear equations. The North American transmission grid is one of the largest nonlinear engineering systems. Most types of power system analysis require the solution in one form or another of this system of nonlinear equations. The applications described below are a handful of the more common applications, but are certainly not a complete coverage of all possible nonlinear problems that arise in power system analysis.

3.6.1 Power Flow

Many power system problems give rise to systems of nonlinear equations that must be solved. Probably the most common nonlinear power system problem is the *power flow* or *load flow* problem. The underlying principle of a power flow problem is that given the system loads, generation, and network configuration, the system bus voltages and line flows can be found by solving the nonlinear power flow equations. This is typically accomplished by applying Kircho 's law at each power system bus throughout the system. In this context, Kircho 's law can be interpreted as *the sum of the powers entering a bus must be zero,* or that the power at each bus must be conserved. Since power is comprised of two components, active power and reactive power, each bus gives rise to two equations – one for active power and one for reactive power. These equations are known as the *power flow equations*:

$$0 = \Delta P_i = P_i^{inj} - V_i \sum_{j=1}^{N_{bus}} V_j Y_{ij} \cos(\theta_i - \theta_j - \phi_{ij}) \tag{3.90}$$

$$0 = \Delta Q_i = Q_i^{inj} - V_i \sum_{j=1}^{N_{bus}} V_j Y_{ij} \sin(\theta_i - \theta_j - \phi_{ij}) \tag{3.91}$$

$$i = 1, \ldots, N_{bus}$$

where P_i^{inj}, Q_i^{inj} are the active and reactive power injected at the bus i, respectively. Loads are modeled by negative power injection. The values V_i and V_j are the voltage magnitudes at bus i and bus j, respectively. The values θ_i and θ_j are the corresponding phase angles. The value $Y_{ij}\angle\phi_{ij}$ is the $(ij)^{th}$ element of the network admittance matrix Y. The constant N_{bus} is the number of buses in the system. The updates ΔP_i^k and ΔQ_i^k of equations (3.90) and (3.91) are called the *mismatch* equations because they give a measure of the power di erence, or mismatch, between the calculated power values, as

functions of voltage and phase angle, and the actual injected powers. As the Newton-Raphson iteration continues, this mismatch is driven to zero until the power leaving a bus, calculated from the voltages and phase angles, equals the injected power. At this point the converged values of voltages and phase angles are used to calculate line flows, swing bus powers, and the injected reactive powers at the generator buses.

The formulation in equations (3.90) and (3.91) is called the *polar* formulation of the power flow equations. If $Y_{ij}\angle\phi_{ij}$ is instead given by the complex sum $g_{ij} + jb_{ij}$, then the power flow equations may be written in *rectangular form* as

$$0 = P_i^{inj} - V_i \sum_{j=1}^{N_{bus}} V_j \left(g_{ij}\cos(\theta_i - \theta_j) + b_{ij}\sin(\theta_i - \theta_j)\right) \tag{3.92}$$

$$0 = Q_i^{inj} - V_i \sum_{j=1}^{N_{bus}} V_j \left(g_{ij}\sin(\theta_i - \theta_j) - b_{ij}\cos(\theta_i - \theta_j)\right) \tag{3.93}$$

$$i = 1, \ldots, N_{bus}$$

In either case, the power flow equations are a system of nonlinear equations. They are nonlinear in both the voltage and phase angle. There are, at most, $2N_{bus}$ equations to solve. This number is then further reduced by removing one power flow equation for each known voltage (at voltage controlled buses) and the swing bus angle. This reduction is necessary since the number of equations must equal the number of unknowns in a fully determined system. Once the nonlinear power flow equations have been determined, the Newton-Raphson method may be directly applied.

The most common approach to solving the power flow equations by the Newton-Raphson method is to arrange the equations by phase angle followed by the voltage magnitudes as

$$\begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} \begin{bmatrix} \Delta\delta_1 \\ \Delta\delta_2 \\ \Delta\delta_3 \\ \vdots \\ \Delta\delta_{N_{bus}} \\ \Delta V_1 \\ \Delta V_2 \\ \Delta V_3 \\ \vdots \\ \Delta V_{N_{bus}} \end{bmatrix} = - \begin{bmatrix} \Delta P_1 \\ \Delta P_2 \\ \Delta P_3 \\ \vdots \\ \Delta P_{N_{bus}} \\ \Delta Q_1 \\ \Delta Q_2 \\ \Delta Q_3 \\ \vdots \\ \Delta Q_{N_{bus}} \end{bmatrix} \tag{3.94}$$

where

$$\Delta\delta_i = \delta_i^{k+1} - \delta_i^k$$
$$\Delta V_i = V_i^{k+1} - V_i^k$$

These equations are then solved using LU factorization with forward/backward substitution. The Jacobian is typically divided into four submatrices, where

$$\begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & \frac{\partial \Delta P}{\partial V} \\ \frac{\partial \Delta Q}{\partial \delta} & \frac{\partial \Delta Q}{\partial V} \end{bmatrix} \tag{3.95}$$

Each submatrix represents the partial derivatives of each of the mismatch equations with respect to each of the unknowns. These partial derivatives yield eight types – two for each mismatch equation, where one is for the diagonal element and the other is for o -diagonal elements. The derivatives are summarized as

$$\frac{\partial \Delta P_i}{\partial \delta_i} = V_i \sum_{j=1}^{N_{bus}} V_j Y_{ij} \sin(\delta_i - \delta_j - \phi_{ij}) + V_i^2 Y_{ii} \sin \phi_{ii} \tag{3.96}$$

$$\frac{\partial \Delta P_i}{\partial \delta_j} = -V_i V_j Y_{ij} \sin(\delta_i - \delta_j - \phi_{ij}) \tag{3.97}$$

$$\frac{\partial \Delta P_i}{\partial V_i} = -\sum_{i=1}^{N_{bus}} V_j Y_{ij} \cos(\delta_i - \delta_j - \phi_{ij}) - V_i Y_{ii} \cos \phi_{ii} \tag{3.98}$$

$$\frac{\partial \Delta P_i}{\partial V_j} = -V_i Y_{ij} \cos(\delta_i - \delta_j - \phi_{ij}) \tag{3.99}$$

$$\frac{\partial \Delta Q_i}{\partial \delta_i} = -V_i \sum_{j=1}^{N_{bus}} V_j Y_{ij} \cos(\delta_i - \delta_j - \phi_{ij}) + V_i^2 Y_{ii} \cos \phi_{ii} \tag{3.100}$$

$$\frac{\partial \Delta Q_i}{\partial \delta_j} = V_i V_j Y_{ij} \cos(\delta_i - \delta_j - \phi_{ij}) \tag{3.101}$$

$$\frac{\partial \Delta Q_i}{\partial V_i} = -\sum_{j=1}^{N_{bus}} V_j Y_{ij} \sin(\delta_i - \delta_j - \phi_{ij}) + V_i Y_{ii} \sin \phi_{ii} \tag{3.102}$$

$$\frac{\partial \Delta Q_i}{\partial V_j} = -V_i Y_{ij} \sin(\delta_i - \delta_j - \phi_{ij}) \tag{3.103}$$

A common modification to the power flow solution is to replace the unknown update ΔV_i by the normalized value $\frac{\Delta V_i}{V_i}$. This formulation yields a more symmetric Jacobian as the Jacobian submatrices J_2 and J_4 are now multiplied by V_i to compensate for the scaling of ΔV_i by V_i. All partial derivatives of each submatrix then become quadratic in voltage magnitude.

The Newton-Raphson method for the solution of the power flow equations is relatively straightforward to program since both the function evaluations and the partial derivatives use the same expressions. Thus it takes little extra computational e ort to compute the Jacobian once the mismatch equations have been calculated.

Example 3.9

Find the voltage magnitudes, phase angles, and line flows for the small power system shown in Figure 3.12 with the following system parameters in per unit:

bus	type	V	P_{gen}	Q_{gen}	P_{load}	Q_{load}
1	swing	1.02	–	–	0.0	0.0
2	PV	1.00	0.5	–	0.0	0.0
3	PQ	–	0.0	0.0	1.2	0.5

i	j	R_{ij}	X_{ij}	B_{ij}
1	2	0.02	0.3	0.15
1	3	0.01	0.1	0.1
2	3	0.01	0.1	0.1

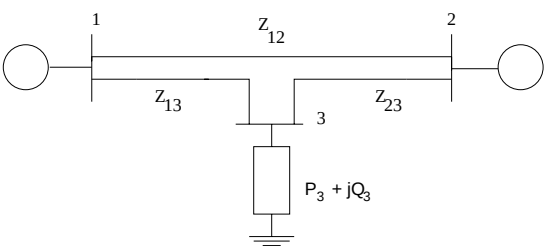


FIGURE 3.12
Example power system

Solution 3.9 The first step in any power flow solution is to calculate the admittance matrix Y for the power system. A simple procedure for calculating the elements of the admittance matrix is

$Y(i,j)$ negative of the admittance between buses i and j
$Y(i,i)$ sum of all admittances connected to bus i

Calculating the admittance matrix for this system yields:

$$Y = \begin{bmatrix} 13.1505\angle -84.7148^\circ & 3.3260\angle 93.8141^\circ & 9.9504\angle 95.7106^\circ \\ 3.3260\angle 95.7106^\circ & 13.1505\angle -84.7148^\circ & 9.9504\angle 95.7106^\circ \\ 9.9504\angle 95.7106^\circ & 9.9504\angle 95.7106^\circ & 19.8012\angle -84.2606^\circ \end{bmatrix} \quad (3.104)$$

By inspection, this system has three unknowns: δ_2, δ_3, and V_3; thus, three power flow equations are required. These power flow equations are

$$0 = \Delta P_2 = 0.5 - V_2 \sum_{j=1}^{3} V_j Y_{ij} \cos(\delta_2 - \delta_j - \theta_{ij}) \tag{3.105}$$

$$0 = \Delta P_3 = -1.2 - V_3 \sum_{j=1}^{3} V_j Y_{ij} \cos(\delta_3 - \delta_j - \theta_{ij}) \tag{3.106}$$

$$0 = \Delta Q_3 = -0.5 - V_3 \sum_{j=1}^{3} V_j Y_{ij} \sin(\delta_3 - \delta_j - \theta_{ij}) \tag{3.107}$$

Substituting in the known quantities for $V_1 = 1.02, V_2 = 1.00$, and $\delta_1 = 0$ and the admittance matrix quantities yields:

$$\begin{aligned} \Delta P_2 = {} & 0.5 - (1.00)\,((1.02)(3.3260)\cos(\delta_2 - 0 - 93.8141^\circ) \\ & + (1.00)(13.1505)\cos(\delta_2 - \delta_2 + 84.7148^\circ) \\ & + (V_3)(9.9504)\cos(\delta_2 - \delta_3 - 95.7106^\circ)) \end{aligned} \tag{3.108}$$

$$\begin{aligned} \Delta P_3 = {} & -1.2 - (V_3)\,((1.02)(9.9504)\cos(\delta_3 - 0 - 95.7106^\circ) \\ & + (1.00)(9.9504)\cos(\delta_3 - \delta_2 - 95.7106^\circ) \\ & + (V_3)(19.8012)\cos(\delta_3 - \delta_3 + 84.2606^\circ)) \end{aligned} \tag{3.109}$$

$$\begin{aligned} \Delta Q_3 = {} & -0.5 - (V_3)\,((1.02)(9.9504)\sin(\delta_3 - 0 - 95.7106^\circ) \\ & + (1.00)(9.9504)\sin(\delta_3 - \delta_2 - 95.7106^\circ) \\ & + ((V_3)(19.8012)\sin(\delta_3 - \delta_3 + 84.2606^\circ)) \end{aligned} \tag{3.110}$$

The Newton-Raphson iteration for this system is then given by

$$\begin{bmatrix} \frac{\partial \Delta P_2}{\partial \delta_2} & \frac{\partial \Delta P_2}{\partial \delta_3} & \frac{\partial \Delta P_2}{\partial V_3} \\ \frac{\partial \Delta P_3}{\partial \delta_2} & \frac{\partial \Delta P_3}{\partial \delta_3} & \frac{\partial \Delta P_3}{\partial V_3} \\ \frac{\partial \Delta Q_3}{\partial \delta_2} & \frac{\partial \Delta Q_3}{\partial \delta_3} & \frac{\partial \Delta Q_3}{\partial V_3} \end{bmatrix} \begin{bmatrix} \Delta\delta_2 \\ \Delta\delta_3 \\ \Delta V_3 \end{bmatrix} = - \begin{bmatrix} \Delta P_2 \\ \Delta P_3 \\ \Delta Q_3 \end{bmatrix} \tag{3.111}$$

where

$$\begin{aligned} \frac{\partial \Delta P_2}{\partial \delta_2} &= 3.3925 \sin(\delta_2 - 93.8141^\circ) \\ &\quad + 9.9504 V_3 \sin(\delta_2 - \delta_3 - 95.7106^\circ) \\ \frac{\partial \Delta P_2}{\partial \delta_3} &= -9.9504 V_3 \sin(\delta_2 - \delta_3 - 95.7106^\circ) \\ \frac{\partial \Delta P_2}{\partial V_3} &= -9.9504 \cos(\delta_2 - \delta_3 - 95.7106^\circ) \\ \frac{\partial \Delta P_3}{\partial \delta_2} &= -9.9504 V_3 \sin(\delta_3 - \delta_2 - 95.7106^\circ) \end{aligned}$$

$$\frac{\partial \Delta P_3}{\partial \delta_3} = 10.1494V_3 \sin(\delta_3 - 95.7106^\circ) + 9.9504V_3 \sin(\delta_3 - \delta_2 - 95.7106^\circ)$$

$$\frac{\partial \Delta P_3}{\partial V_3} = -10.1494 \cos(\delta_3 - 95.7106^\circ) - 9.9504 \cos(\delta_3 - \delta_2 - 95.7106^\circ) - 39.6024V_3 \cos(84.2606^\circ)$$

$$\frac{\partial \Delta Q_3}{\partial \delta_2} = 9.9504V_3 \cos(\delta_3 - \delta_2 - 95.7106^\circ)$$

$$\frac{\partial \Delta Q_3}{\partial \delta_3} = -10.1494V_3 \cos(\delta_3 - 95.7106^\circ) - 9.9504V_3 \cos(\delta_3 - \delta_2 - 95.7106^\circ)$$

$$\frac{\partial \Delta Q_3}{\partial V_3} = -10.1494 \sin(\delta_3 - 95.7106^\circ) - 9.9504 \sin(\delta_3 - \delta_2 - 95.7106^\circ) - 39.6024V_3 \sin(84.2606^\circ)$$

Recall that one of the underlying assumptions of the Newton-Raphson iteration is that the higher order terms of the Taylor series expansion are negligible only if the initial guess is su ciently close to the actual solution to the nonlinear equations. Under most operating conditions, the voltages throughout the power system are within ±10% of the nominal voltage and therefore fall in the range $0.9 \leq V_i \leq 1.1$ per unit. Similarly, under most operating conditions the phase angle di erences between adjacent buses are typically small. Thus if the swing bus angle is taken to be zero, then all phase angles throughout the system will also be close to zero. Therefore in initializing a power flow, it is common to choose a "flat start" initial condition. That is, all voltage magnitudes are set to 1.0 per unit and all angles are set to zero.

Iteration 1

Evaluating the Jacobian and the mismatch equations at the flat start initial conditions yields:

$$[J^0] = \begin{bmatrix} -13.2859 & 9.9010 & 0.9901 \\ 9.9010 & -20.0000 & -1.9604 \\ -0.9901 & 2.0000 & -19.4040 \end{bmatrix}$$

$$\begin{bmatrix} \Delta P_2^0 \\ \Delta P_3^0 \\ \Delta Q_3^0 \end{bmatrix} = \begin{bmatrix} 0.5044 \\ -1.1802 \\ -0.2020 \end{bmatrix}$$

Solving

$$[J^0] \begin{bmatrix} \Delta \delta_2^1 \\ \Delta \delta_3^1 \\ \Delta V_3^1 \end{bmatrix} = - \begin{bmatrix} \Delta P_2^0 \\ \Delta P_3^0 \\ \Delta Q_3^0 \end{bmatrix}$$

by LU factorization yields:

$$\begin{bmatrix} \Delta\delta_2^1 \\ \Delta\delta_3^1 \\ \Delta V_3^1 \end{bmatrix} = \begin{bmatrix} -0.0096 \\ -0.0621 \\ -0.0163 \end{bmatrix}$$

Therefore

$$\delta_2^1 = \delta_2^0 + \Delta\delta_2^1 = 0 - 0.0096 = -0.0096$$
$$\delta_3^1 = \delta_3^0 + \Delta\delta_3^1 = 0 - 0.0621 = -0.0621$$
$$V_3^1 = V_3^0 + \Delta V_3^1 = 1 - 0.0163 = 0.9837$$

Note that the angles are given in *radians* and not degrees. The error at the first iteration is the largest absolute value of the mismatch equations, which is

$$\varepsilon^1 = 1.1802$$

One quick check of this process is to note that the voltage update V_3^1 is slightly less than 1.0 per unit, which would be expected given the system configuration. Note also that the diagonals of the Jacobian are all equal or greater in magnitude than the o -diagonal elements. This is because the diagonals are summations of terms, whereas the o -diagonal elements are single terms.

Iteration 2

Evaluating the Jacobian and the mismatch equations at the updated values δ_2^1, δ_3^1, and V_3^1 yields:

$$\left[J^1\right] = \begin{bmatrix} -13.1597 & 9.7771 & 0.4684 \\ 9.6747 & -19.5280 & -0.7515 \\ -1.4845 & 3.0929 & -18.9086 \end{bmatrix}$$

$$\begin{bmatrix} \Delta P_2^1 \\ \Delta P_3^1 \\ \Delta Q_3^1 \end{bmatrix} = \begin{bmatrix} 0.0074 \\ -0.0232 \\ -0.0359 \end{bmatrix}$$

Solving for the update yields

$$\begin{bmatrix} \Delta\delta_2^2 \\ \Delta\delta_3^2 \\ \Delta V_3^2 \end{bmatrix} = \begin{bmatrix} -0.0005 \\ -0.0014 \\ -0.0021 \end{bmatrix}$$

and

$$\begin{bmatrix} \delta_2^2 \\ \delta_3^2 \\ V_3^2 \end{bmatrix} = \begin{bmatrix} -0.0101 \\ -0.0635 \\ 0.9816 \end{bmatrix}$$

where

$$\varepsilon^2 = 0.0359$$

Iteration 3

Evaluating the Jacobian and the mismatch equations at the updated values δ_2^2, δ_3^2, and V_3^2 yields:

$$\left[J^2\right] = \begin{bmatrix} -13.1392 & 9.7567 & 0.4600 \\ 9.6530 & -19.4831 & -0.7213 \\ -1.4894 & 3.1079 & -18.8300 \end{bmatrix}$$

$$\begin{bmatrix} \Delta P_2^0 \\ \Delta P_3^0 \\ \Delta Q_3^0 \end{bmatrix} = \begin{bmatrix} 0.1717 \\ -0.5639 \\ -0.9084 \end{bmatrix} \times 10^{-4}$$

Solving for the update yields

$$\begin{bmatrix} \Delta\delta_2^2 \\ \Delta\delta_3^2 \\ \Delta V_3^2 \end{bmatrix} = \begin{bmatrix} -0.1396 \\ -0.3390 \\ -0.5273 \end{bmatrix} \times 10^{-5}$$

and

$$\begin{bmatrix} \delta_2^3 \\ \delta_3^3 \\ V_3^3 \end{bmatrix} = \begin{bmatrix} -0.0101 \\ -0.0635 \\ 0.9816 \end{bmatrix}$$

where

$$\varepsilon^3 = 0.9084 \times 10^{-4}$$

At this point, the iterations have converged since the mismatch is su ciently small and the values are no longer changing significantly.

The last task in power flow is to calculate the generated reactive powers, the swing bus active power output and the line flows. The generated powers can be calculated directly from the power flow equations:

$$P_i^{inj} = V_i \sum_{j=1}^{N_{bus}} V_j Y_{ij} \cos\left(\theta_i - \theta_j - \phi_{ij}\right)$$

$$Q_i^{inj} = V_i \sum_{j=1}^{N_{bus}} V_j Y_{ij} \sin\left(\theta_i - \theta_j - \phi_{ij}\right)$$

Therefore

$$P_{gen,1} = P_1^{inj} = 0.7087$$
$$Q_{gen,1} = Q_1^{inj} = 0.2806$$
$$Q_{gen,2} = Q_2^{inj} = -0.0446$$

The active power losses in the system are the di erence between the sum of the generation and the sum of the loads, in this case:

$$P_{loss} = \sum P_{gen} - \sum P_{load} = 0.7087 + 0.5 - 1.2 = 0.0087\ pu \qquad (3.112)$$

The line losses for line $i-j$ are calculated at both the sending and receiving ends of the line. Therefore the power sent from bus i to bus j is

$$S_{ij} = V_i \angle \delta_i I_{ij}^* \tag{3.113}$$

and the power received at bus j from bus i is

$$S_{ji} = V_j \angle \delta_j I_{ji}^* \tag{3.114}$$

Thus

$$P_{ij} = V_i V_j Y_{ij} \cos(\delta_i - \delta_j - \phi_{ij}) - V_i^2 Y_{ij} \cos(\phi_{ij}) \tag{3.115}$$

$$Q_{ij} = V_i V_j Y_{ij} \sin(\delta_i - \delta_j - \phi_{ij}) + V_i^2 Y_{ij} \sin(\phi_{ij}) \tag{3.116}$$

Similarly, the powers P_{ji} and Q_{ji} can be calculated. The active power loss on any given line is the di erence between the active power sent from bus i and the active power received at bus j. Calculating the reactive power losses is more complex since the reactive power generated by the line-charging (shunt capacitances) must also be included.

3.6.2 Regulating Transformers

One of the most common controllers found in the power system network is the *regulating transformer*. This is a transformer that is able to change the winding ratios (tap settings) in response to changes in load-side voltage. If the voltage on the secondary side (or load side) is lower than a desired voltage (such as during heavy loading), the tap will change so as to increase the secondary voltage while maintaining the primary side voltage. A regulating transformer is also frequently referred to as an *under-load-tap-changing* or ULTC transformer. The tap setting t may be real or complex, and in per unit, the tap ratio is defined as $1 : t$ where t is typically within 10% of 1.0. A phase-shifting transformer is achieved by allowing the tap t to be complex with both magnitude and angle.

The e ect of the regulating transformer is incorporated into the power flow algorithm through the admittance matrix. To incorporate a regulating transformer into the admittance matrix, consider the regulating transformer as a two-port network relating the input currents I_i and I_j to the input voltages V_i and V_j as shown in Figure 3.13. The receiving end current is given by

$$I_j = (V_j - tV_i)\, Y \tag{3.117}$$

Note that the currents can be found from the power transfer equation:

$$S_i = V_i I_i^* = -tV_i I_j^* \tag{3.118}$$

Therefore

$$I_i = -t^* I_j \tag{3.119}$$

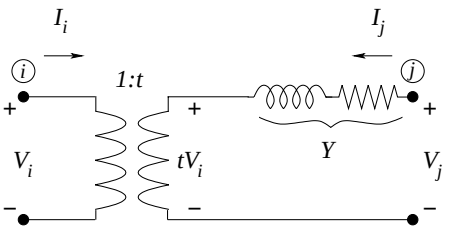


FIGURE 3.13
A regulating transformer

$$= -t^* (V_j - tV_i) Y \tag{3.120}$$
$$= tt^* Y V_i - t^* Y V_j \tag{3.121}$$
$$= |t|^2 Y V_i - t^* Y V_j \tag{3.122}$$

Therefore the o -diagonal entries in the admittance matrix become:

$$Y(i,j) = -t^* Y$$
$$Y(j,i) = -tY$$

and $|t|^2 Y$ is added to $Y(i,i)$ and Y is added to $Y(j,j)$.

Since regulating transformers are used as voltage control devices, a common computational exercise is to find the tap setting t that will hold the secondary bus voltage magnitude V_j at a specified voltage $\hat{V}$. This may be interpreted as adding one additional variable to the system (t) and one additional constraint $\left(V_j = \hat{V}\right)$. Since the additional constraint is counterbalanced by the additional degree of freedom, the dimension of the problem remains the same. There are two primary approaches for finding the tap setting t that results in $V_j = \hat{V}$. One approach is an iterative approach while the second approach calculates t directly from the power flow equations.

The iterative approach may be summarized as:

1. Set $t = t_0$
2. Run a power flow to calculate V_j
3. Is $V_j > \hat{V}$? If yes, then $t = t - \Delta t$, and go to step 2.
4. Is $V_j < \hat{V}$? If yes, then $t = t + \Delta t$, and go to step 2.
5. Done

This approach is conceptually simple and requires no changes to the power flow algorithm. However, it may require numerous runs of a power flow program if t_0 is far from the required tap setting.

The direct approach applies the Newton-Raphson method directly to the updated power flow equations as functions of the tap setting t.

1. Set $V_j = \hat{V}$ and let t be an unknown state
2. Modify the Newton-Raphson Jacobian such that the row of partial derivatives with respect to V_j are replaced by the row of partial derivatives with respect to t
3. Modify the state vector x such that

$$x = \begin{bmatrix} \delta_2 \\ \delta_3 \\ \vdots \\ \delta_n \\ V_2 \\ V_3 \\ \vdots \\ V_{j-1} \\ t \\ V_{j+1} \\ \vdots \\ V_n \end{bmatrix}$$

   Note that the state V_j is replaced by t.

4. Perform the Newton-Raphson

In this case, the set of power flow equations is solved only once, but since the system Jacobian is modified, a standard power flow program cannot be used.

Since the tap cannot move continuously along the transformer windings, but must move vertically from one winding to the adjacent winding, the real tap setting is not a continuous state. Therefore, in both cases, the calculated tap setting must be rounded to the nearest possible physical tap setting.

Example 3.10

For the system shown in Figure 3.12, place a transformer with reactance X and real tap t between bus 3 and the load (introduce a new bus 4). Find the new admittance matrix and the corresponding Jacobian entries.

Solution 3.10 Let the admittance matrix of the subsystem containing buses 1-3 be given by:

$$Y_{bus} = \begin{bmatrix} Y_{11}\angle\theta_{11} & Y_{12}\angle\theta_{12} & Y_{13}\angle\theta_{13} \\ Y_{21}\angle\theta_{21} & Y_{22}\angle\theta_{22} & Y_{23}\angle\theta_{23} \\ Y_{31}\angle\theta_{31} & Y_{32}\angle\theta_{32} & Y_{33}\angle\theta_{33} \end{bmatrix} \tag{3.123}$$

Adding the transformer between buses 3 and 4 yields the new admittance matrix:

$$Y_{bus} = \begin{bmatrix} Y_{11}\angle\theta_{11} & Y_{12}\angle\theta_{12} & Y_{13}\angle\theta_{13} & 0 \\ Y_{21}\angle\theta_{21} & Y_{22}\angle\theta_{22} & Y_{23}\angle\theta_{23} & 0 \\ Y_{31}\angle\theta_{31} & Y_{32}\angle\theta_{32} & Y_{33}\angle\theta_{33} + \frac{t^2}{jX} & \frac{-t}{jX} \\ 0 & 0 & \frac{-t}{jX} & \frac{1}{jX} \end{bmatrix} \tag{3.124}$$

The power flow equations at bus 3 become:

$$\begin{aligned} 0 &= P_3 - V_3V_1Y_{31}\cos(\delta_3 - \delta_1 - \theta_{31}) - V_3V_2Y_{32}\cos(\delta_3 - \delta_2 - \theta_{32}) \\ &\quad -V_3V_4\begin{pmatrix} t \\ X \end{pmatrix}\cos(\delta_3 - \delta_4 - 90^\circ) - V_3^2Y_{33}\cos(-\theta_{33}) - V_3^2\begin{pmatrix} t^2 \\ X \end{pmatrix}\cos(90^\circ) \\ 0 &= Q_3 - V_3V_1Y_{31}\sin(\delta_3 - \delta_1 - \theta_{31}) - V_3V_2Y_{32}\sin(\delta_3 - \delta_2 - \theta_{32}) \\ &\quad -V_3V_4\begin{pmatrix} t \\ X \end{pmatrix}\sin(\delta_3 - \delta_4 - 90^\circ) - V_3^2Y_{33}\sin(-\theta_{33}) - V_3^2\begin{pmatrix} t^2 \\ X \end{pmatrix}\sin(90^\circ) \end{aligned}$$

Since V_4 is specified, there is no partial derivative $\frac{\partial \Delta P_3}{\partial V_4}$; instead there is a partial derivative with respect to t:

$$\frac{\partial \Delta P_3}{\partial t} = -\frac{V_3V_4}{X}\cos(\delta_3 - \delta_4 - 90^\circ) \tag{3.125}$$

Similarly, the partial derivative of $\frac{\partial \Delta Q_3}{\partial t}$ becomes

$$\frac{\partial \Delta Q_3}{\partial t} = -\frac{V_3V_4}{X}\sin(\delta_3 - \delta_4 - 90^\circ) + 2V_3^2\frac{t}{X} \tag{3.126}$$

The partial derivatives with respect to δ_1, δ_2, V_1, and V_2 do not change, but the partial derivatives with respect to δ_3, δ_4, and V_3 become

$$\begin{aligned} \frac{\partial \Delta P_3}{\partial \delta_3} &= V_3V_1Y_{31}\sin(\delta_3 - \delta_1 - \theta_{31}) + V_3V_2Y_{32}\sin(\delta_3 - \delta_2 - \theta_{32}) \\ &\quad + V_3V_4\frac{t}{X}\sin(\delta_3 - \delta_4 - 90^\circ) \\ \frac{\partial \Delta P_3}{\delta_4} &= -V_3V_4\frac{t}{X}\sin(\delta_3 - \delta_4 - 90^\circ) \\ \frac{\partial \Delta P_3}{\partial V_3} &= -V_1Y_{31}\cos(\delta_3 - \delta_1 - \theta_{31}) - V_2Y_{32}\cos(\delta_3 - \delta_2 - \theta_{32}) \\ &\quad V_4\frac{t}{X}\cos(\delta_3 - \delta_4 - 90^\circ) - 2V_3Y_{33}\cos(-\theta_{33}) \\ \frac{\partial \Delta Q_3}{\partial \delta_3} &= -V_3V_1Y_{31}\cos(\delta_3 - \delta_1 - \theta_{31}) - V_3V_2Y_{32}\cos(\delta_3 - \delta_2 - \theta_{32}) \\ &\quad -V_3V_4\frac{t}{X}\cos(\delta_3 - \delta_4 - 90^\circ) \\ \frac{\partial \Delta Q_3}{\partial \delta_4} &= V_3V_4\frac{t}{X}\cos(\delta_3 - \delta_4 - 90^\circ) \end{aligned}$$

$$\frac{\partial \Delta Q_3}{\partial V_3} = -V_1 Y_{31} \sin(\delta_3 - \delta_1 - \theta_{31}) - V_2 Y_{32} \sin(\delta_3 - \delta_2 - \theta_{32})$$
$$-V_4 \frac{t}{X} \sin(\delta_3 - \delta_4 - 90^\circ) - 2V_3 Y_{33} \sin(-\theta_{33}) - 2V_3 \frac{t^2}{X}$$

These partial derivatives are used in developing the Newton-Raphson Jacobian for the iterative power flow method.

3.6.3 Decoupled Power Flow

The power flow is one of the most widely used computational tools in power systems analysis. It can be successfully applied to problems ranging from a single machine system to a power system containing tens of thousands of buses. For very large systems, the full power flow may require significant computational resources to calculate, store, and factorize the Jacobian matrix. As discussed previously, however, it is possible to replace the Jacobian matrix with a matrix M that is easier to calculate and factor and still retain good convergence properties. The power flow equations naturally lend themselves to several alternate matrices for the power flow solution that can be derived from the formulation of the system Jacobian. Recall that the system Jacobian has the form:

$$\begin{bmatrix} J_1 & J_2 \\ J_3 & J_4 \end{bmatrix} = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & \frac{\partial \Delta P}{\partial V} \\ \frac{\partial \Delta Q}{\partial \delta} & \frac{\partial \Delta Q}{\partial V} \end{bmatrix} \tag{3.127}$$

The general form of the P submatrices are:

$$\frac{\partial \Delta P_i}{\partial \delta_j} = -V_i V_j Y_{ij} \sin(\delta_i - \delta_j - \phi_{ij}) \tag{3.128}$$

$$\frac{\partial \Delta P_i}{\partial V_j} = V_i Y_{ij} \cos(\delta_i - \delta_j - \phi_{ij}) \tag{3.129}$$

For most transmission lines, the line resistance contributes only nominally to the overall line impedance; thus, the phase angles ϕ_{ij} of the admittance matrix entries are near $\pm 90^\circ$. Additionally, under normal operating conditions the phase angle di erence between adjacent buses is typically small; therefore:

$$\cos(\delta_i - \delta_j - \phi_{ij}) \approx 0 \tag{3.130}$$

leading to

$$\frac{\partial \Delta P_i}{\partial V_j} \approx 0 \tag{3.131}$$

Similar arguments can be made such that

$$\frac{\partial \Delta Q_i}{\partial \delta_j} \approx 0 \tag{3.132}$$

Using the approximations of equations (3.131) and (3.132), a possible substitution for the Jacobian matrix is the matrix

$$M = \begin{bmatrix} \frac{\partial \Delta P}{\partial \delta} & 0 \\ 0 & \frac{\partial \Delta Q}{\partial V} \end{bmatrix} \tag{3.133}$$

Using this matrix M as a replacement for the system Jacobian leads to a set of *decoupled* iterates for the power flow solution:

$$\delta^{k+1} = \delta^k - \left[\frac{\partial \Delta P}{\partial \delta}\right]^{-1} \Delta P \tag{3.134}$$

$$V^{k+1} = V^k - \left[\frac{\partial \Delta Q}{\partial V}\right]^{-1} \Delta Q \tag{3.135}$$

where the ΔP and ΔQ iterations can be carried out independently. The primary advantage of this decoupled power flow is that the LU factorization computation is significantly reduced. The LU factorization of the full Jacobian requires $(2n)^3 = 8n^3$ floating point operations per iteration, whereas the decoupled power flow requires only $2n^3$ floating point operations per iteration.

Example 3.11

Repeat Example 3.9 using the decoupled power flow algorithm.

Solution 3.11 The Jacobian of Example 3.9 evaluated at the initial condition is

$$[J^0] = \begin{bmatrix} -13.2859 & 9.9010 & 0.9901 \\ 9.9010 & -20.0000 & -1.9604 \\ -0.9901 & 2.0000 & -19.4040 \end{bmatrix} \tag{3.136}$$

Note that the o -diagonal submatrices are much smaller in magnitude than the diagonal submatrices. For example,

$$[J_2] = \left\| \begin{bmatrix} 0.9901 \\ -1.9604 \end{bmatrix} \right\| << [J_1] = \left\| \begin{bmatrix} -13.2859 & 9.9010 \\ 9.9010 & -20.000 \end{bmatrix} \right\|$$

and

$$[J_3] = \left\| \begin{bmatrix} -0.9901 & 2.0000 \end{bmatrix} \right\| << [J_4] = [-19.4040]$$

Thus, it is reasonable to neglect the o -diagonal matrices J_2 and J_3. Therefore, the first iteration of the decoupled power flow becomes:

$$\begin{bmatrix} \Delta\delta_2^1 \\ \Delta\delta_3^1 \end{bmatrix} = [J_1]^{-1} \begin{bmatrix} \Delta P_2 \\ \Delta P_3 \end{bmatrix} \tag{3.137}$$

$$= \begin{bmatrix} -13.2859 & 9.9010 \\ 9.9010 & -20.000 \end{bmatrix}^{-1} \begin{bmatrix} 0.5044 \\ -1.1802 \end{bmatrix} \tag{3.138}$$

$$\left[\Delta V_3^1\right] = [J_4]^{-1} \Delta Q_3 \tag{3.139}$$

$$= -19.4040^{-1}(-0.2020) \tag{3.140}$$

leading to the updates

$$\begin{bmatrix} \delta_2^1 \\ \delta_3^1 \\ V_3^1 \end{bmatrix} = \begin{bmatrix} -0.0095 \\ -0.0637 \\ 0.9896 \end{bmatrix}$$

The iterative process continues similar to the full Newton-Raphson method by continually updating the J_1 and J_4 Jacobian submatrices and the mismatch equations. The iteration converges when both the ΔP mismatch equations and the ΔQ mismatch equations are both less than the convergence tolerance. Note that it is possible for one set of mismatch equations to meet the convergence criteria before the other; thus, the number of "P" iterations required for convergence may di er from the number of "Q" iterations required for convergence.

3.6.4 Fast Decoupled Power Flow

In Example 3.11, each of the decoupled Jacobian submatrices is updated at every iteration. As discussed previously, however, it is often desirable to have constant matrices to minimize the number of function evaluations and LU factorizations. This is often referred to as the *fast decoupled power flow* and can be represented as:

$$\left[\Delta P^k\right] = [B'] \left[\Delta \delta^{k+1}\right] \tag{3.141}$$

$$\begin{bmatrix} \Delta Q^k \\ V \end{bmatrix} = [B''] \left[\Delta V^{k+1}\right] \tag{3.142}$$

where the B' and B'' are constant [48]. To derive these matrices from the power flow Jacobian, consider the decoupled power flow relationships for the Newton-Raphson method:

$$[\Delta P] = -[J_1][\Delta \delta] \tag{3.143}$$

$$\begin{bmatrix} \Delta Q \\ V \end{bmatrix} = -[J_4][\Delta V] \tag{3.144}$$

where the Jacobian submatrices in rectangular form are:

$$J_1(i,i) = V_i \sum_{j \neq i} V_j \left(g_{ij} \sin \delta_{ij} - b_{ij} \cos \delta_{ij}\right) \tag{3.145}$$

$$J_1(i,j) = -V_i V_j \left(g_{ij} \sin \delta_{ij} - b_{ij} \cos \delta_{ij}\right) \tag{3.146}$$

$$J_4(i,i) = 2V_i b_{ii} - \sum_{j \neq i} V_j \left(g_{ij} \sin \delta_{ij} - b_{ij} \cos \delta_{ij}\right) \tag{3.147}$$

$$J_4(i,j) = -V_i \left(g_{ij} \sin \delta_{ij} - b_{ij} \cos \delta_{ij}\right) \tag{3.148}$$

where $b_{ij} = |Y_{ij} \sin \phi_{ij}|$ are the imaginary elements of the admittance matrix and $g_{ij} = |Y_{ij} \cos \phi_{ij}|$ are the real elements of the admittance matrix. By

noting that $\phi_{ij} \approx 90°$, then $\cos\phi_{ij} \approx 0$ which implies that $g_{ij} \approx 0$. By further approximating all voltage magnitudes as 1.0 per unit, then

$$J_1(i,i) = -\sum_{j\neq i} b_{ij} \tag{3.149}$$

$$J_1(i,j) = b_{ij} \tag{3.150}$$

$$J_4(i,i) = 2b_{ii} - \sum_{j\neq i} b_{ij} \tag{3.151}$$

$$J_4(i,j) = b_{ij} \tag{3.152}$$

Since the J_1 submatrix relates the changes in active power to changes in angle, elements that a ect mainly reactive power flow can be omitted from this matrix with negligible impact on the convergence properties. Thus, shunt capacitors (including line-charging) and external reactances as well as the shunts formed due to representation of o -nominal non-phase-shifting transformers (i.e., taps are set to 1.0) are neglected. Hence, the admittance matrix diagonal elements are devoid of these shunts. Additionally, the lumped series resistances of the transmission lines are also omitted. The resulting approximate matrix B' to the submatrix J_1 is given by

$$B'_{ij} = \frac{1}{x_{ij}} \tag{3.153}$$

$$B'_{ii} = -\sum_{j\neq i} B'_{ij} \tag{3.154}$$

Similarly, the J_4 submatrix relates the changes in reactive power to changes in voltage magnitude; therefore elements that primarily a ect active power flow are omitted. Thus all phase-shifting transformers are neglected, resulting in

$$B''_{ij} = b_{ij} \tag{3.155}$$

$$B''_{ii} = 2b_i - \sum_{j\neq i} B''_{ij} \tag{3.156}$$

where b_i is the shunt susceptance at bus i (i.e., the sum of susceptances of all the shunt branches connected to bus i).

This method results in a set of constant matrices that can be used to approximate the power flow Jacobian in the Newton-Raphson iteration. This method is often referred to as the XB version of the fast decoupled power flow. Both B' and B'' are real, sparse, and contain only network or admittance matrix elements. In the Newton-Raphson method, these matrices are only factorized once for the LU factorization, and are then stored and held constant throughout the iterative solution process. These matrices were derived based on the application of certain assumptions. If these assumptions do not hold (i.e., the voltage magnitudes deviate substantially from 1.0 per unit;

the network has high R/X ratios; or the angle differences between adjacent buses are not small), then convergence problems with the fast decoupled power flow iterations can arise. Work still continues on developing modifications to the XB method to improve convergence [32], [33], [38].

Example 3.12

Repeat Example 3.9 using the fast decoupled power flow algorithm.

Solution 3.12 The line data for the example system are repeated below for convenience:

i	j	R_{ij}	X_{ij}	B_{ij}
1	2	0.02	0.3	0.15
1	3	0.01	0.1	0.1
2	3	0.01	0.1	0.1

and lead to the following admittance matrix:

$$Y_{bus} = \begin{bmatrix} 13.1505\angle -84.7148^\circ & 3.3260\angle 93.8141^\circ & 9.9504\angle 95.7106^\circ \\ 3.3260\angle 95.7106^\circ & 13.1505\angle -84.7148^\circ & 9.9504\angle 95.7106^\circ \\ 9.9504\angle 95.7106^\circ & 9.9504\angle 95.7106^\circ & 19.8012\angle -84.2606^\circ \end{bmatrix} \tag{3.157}$$

Taking the imaginary part of this matrix yields the following B matrix:

$$B = \begin{bmatrix} -13.0946 & 3.3186 & 9.9010 \\ 3.3186 & -13.0946 & 9.9010 \\ 9.9010 & 9.9010 & -19.7020 \end{bmatrix} \tag{3.158}$$

From the line data and the associated B matrix, the following B' and B'' matrices result:

$$B' = \begin{bmatrix} -\frac{1}{x_{21}} - \frac{1}{x_{23}} & \frac{1}{x_{23}} \\ \frac{1}{x_{23}} & -\frac{1}{x_{31}} - \frac{1}{x_{32}} \end{bmatrix} = \begin{bmatrix} -13.3333 & 10 \\ 10 & -20 \end{bmatrix} \tag{3.159}$$

$$\begin{aligned} B'' &= [2b_3 - (B_{31} + B_{32})] \\ &= [2\,(0.05 + 0.05)) - (9.9010 + 9.9010)] = -19.6020 \end{aligned} \tag{3.160}$$

Compare these matrices to the J_1 and J_4 submatrices of Example 3.9 evaluated at the initial condition:

$$J_1 = \begin{bmatrix} -13.2859 & 9.9010 \\ 9.9010 & -20.000 \end{bmatrix}$$

$$J_4 = [-19.4040]$$

The similarity between the matrices is to be expected as a result of the defining assumptions of the fast decoupled power flow method.

Iteration 1

The updates can be found by solving the following linear set of equations

$$\begin{bmatrix} \Delta P_2^0 \\ \Delta P_3^0 \end{bmatrix} = \begin{bmatrix} 0.5044 \\ -1.1802 \end{bmatrix} = - \begin{bmatrix} -13.3333 & 10 \\ 10 & -20 \end{bmatrix} \begin{bmatrix} \Delta \delta_2^1 \\ \Delta \delta_3^1 \end{bmatrix}$$
$$\begin{bmatrix} \Delta Q_3^0 \end{bmatrix} = [-0.2020] = -19.6020 \Delta V_3^1$$

where $\Delta \delta_2^1 = \delta_2^{(1)} - \delta_2^{(0)}$, $\Delta \delta_3^1 = \delta_3^{(1)} - \delta_3^{(0)}$, and $\Delta V_3^1 = V_3^{(1)} - V_3^{(0)}$ and the initial conditions are a "flat start." Solving for the updates yields

$$\begin{bmatrix} \delta_2^1 \\ \delta_3^1 \\ V_3^1 \end{bmatrix} = \begin{bmatrix} -0.0103 \\ -0.0642 \\ 0.9897 \end{bmatrix}$$

where the phase angles are in radians. This process is continued until convergence in both the "P" and "Q" iterations is achieved.

Note that in both the decoupled power flow cases that the objective of the iterations are the same as for the full Newton-Raphson power flow algorithm. The objective is to drive the mismatch equations ΔP and ΔQ to within some tolerance. Therefore, regardless of the number of iterations required to achieve convergence, the accuracy of the answer is the same as for the full Newton-Raphson method. In other words, the voltages and angles of the decoupled power flow methods will be the same as with the full Newton-Raphson method as long as the iterates converge.

3.6.5 PV Curves and Continuation Power Flow

The power flow is a useful tool for monitoring system voltages as a function of load change. One common application is to plot the voltage at a particular bus as the load is varied from the base case to a loadability limit (often known as the point of maximum loadability). If the load is increased to the loadability limit and then decreased back to the original loading, it is possible to trace the entire power-voltage or "PV" curve. This curve, shown in Figure 3.14, is sometimes called the *nose curve* for its shape.

At the loadability limit, or tip of the nose curve, the system Jacobian of the power flow equations will become singular as the slope of the nose curve becomes infinite. Thus, the traditional Newton-Raphson method of obtaining the load flow solution will break down. In this case, a modification of the Newton-Raphson method known as the *continuation method* is employed. The continuation method introduces an additional equation and unknown into the basic power flow equations. The additional equation is chosen specifically to ensure that the augmented Jacobian is no longer singular at the loadability limit. The additional unknown is often called the continuation parameter.

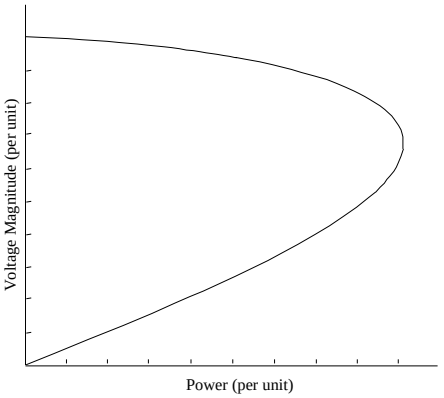


FIGURE 3.14
A PV curve

Continuation methods usually depend on a predictor-corrector scheme and the means to change the continuation parameter as necessary. The basic approach to tracing the PV curve is to choose a new value for the continuation parameter (either in power or voltage) and then predict the power flow solution for this value. This is frequently accomplished using a tangential (or linear) approximation. Using the predicted value as the initial condition for the nonlinear iteration, the augmented power flow equations are then solved (or corrected) to achieve the solution. So the solution is first predicted, and then corrected. This prediction/correction step is shown in Figure 3.15.

Let the set of power flow equations be given as

$$\lambda K - f(\delta, V) = 0 \tag{3.161}$$

or

$$F(\delta, V, \lambda) = 0 \tag{3.162}$$

where K is the loading profile (i.e., the base case relationship between P and Q) and λ is the loading parameter which will vary from unity (at the base case) to the point of maximum loadability. Equation (3.162) may be linearized to yield:

$$\frac{\partial F}{\partial \delta} d\delta + \frac{\partial F}{\partial V} dV + \frac{\partial F}{\partial \lambda} d\lambda = 0 \tag{3.163}$$

Equation (3.163) has one more unknown (λ) than equations, so one more equation is required:

$$e_k \begin{bmatrix} d\delta \\ dV \\ d\lambda \end{bmatrix} = 1 \tag{3.164}$$

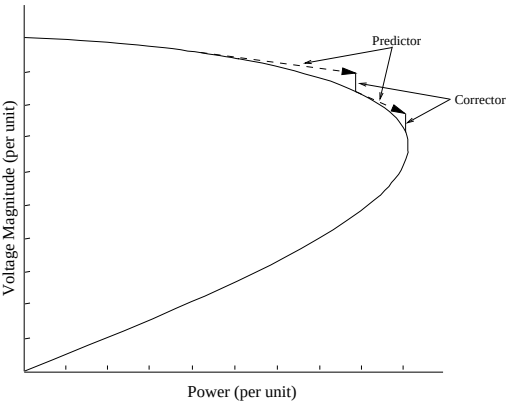


FIGURE 3.15
The predictor/corrector step

where e_k is a row vector of zeros with a single ± 1 at the position of the unknown that is chosen to be the continuation parameter. The sign of the one in e_k is chosen based on whether the continuation parameter is increasing or decreasing. When the continuation parameter is λ (power), the sign is positive indicating that the load is increasing. When voltage is the continuation parameter, the sign is negative, indicating that the voltage magnitude is expected to decrease towards the tip of the nose curve.

The unknowns are predicted such that

$$\begin{bmatrix} \delta \\ V \\ \lambda \end{bmatrix}^{\text{predicted}} = \begin{bmatrix} \delta_0 \\ V_0 \\ \lambda_0 \end{bmatrix} + \sigma \begin{bmatrix} d\delta \\ dV \\ d\lambda \end{bmatrix} \tag{3.165}$$

where

$$\begin{bmatrix} d\delta \\ \\ dV \\ \cdots \\ d\lambda \end{bmatrix} = \begin{bmatrix} & & \vdots & \\ & J_{LF} & \vdots & K \\ & & \vdots & \\ \cdots & \cdots & \cdots & \cdots \\ & & [e_k] & \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

and σ is the step-size (or length) for the next prediction. Note that the

continuation state $dx_k = 1$, thus

$$x_k^{\text{predicted}} = x_{k0} + \sigma$$

so σ should be chosen to represent a reasonable step-size in terms of what the continuation parameter is (usually voltage or power).

The corrector step involves the solution of the set of equations:

$$F(\delta, V, \lambda) = 0 \tag{3.166}$$

$$x_k - x_k^{\text{predicted}} = 0 \tag{3.167}$$

where x_k is the chosen continuation parameter. Typically the continuation parameter is chosen as the state that exhibits the greatest rate of change.

Example 3.13

Plot the PV curve (P versus V) of the system shown in Figure 3.16 using the continuation power flow method as the load varies from zero to the point of maximum loadability.

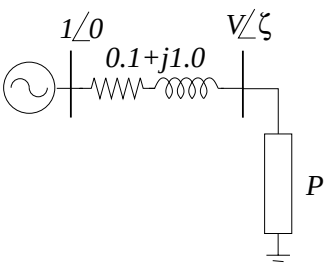


FIGURE 3.16
System for Example 3.13

Solution 3.13 The power flow equations for the system shown in Figure 3.16 are:

$$0 = -P - 0.995V\cos(\delta - 95.7^\circ) - 0.995V^2\cos(84.3^\circ) \tag{3.168}$$

$$0 = -0.995V\sin(\delta - 95.7^\circ) - 0.995V^2\sin(84.3^\circ) \tag{3.169}$$

During the continuation power flow, the vector of injected active and reactive

powers will be replaced by the vector λK. The loading vector λK is

$$\lambda K = \lambda \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

where λ will vary from zero to the maximum loading value. Typically the vector K will contain the base case values for all injected active and reactive powers in the system. In this case, the entry for the load P is negative indicating that the injected power is negative (i.e., a load).

The loadflow Jacobian for this set of power flow equations is

$$J_{LF} = \begin{bmatrix} 0.995V\sin(\delta - 95.7^\circ) & -0.995\cos(\delta - 95.7^\circ) - 1.99\cos(84.3^\circ)\,V \\ -0.995V\cos(\delta - 95.7^\circ) & -0.995\sin(\delta - 95.7^\circ) - 1.99\sin(84.3^\circ)\,V \end{bmatrix}$$

Iteration 1

Initially, the continuation parameter is chosen to be λ since the load will change more rapidly than the voltage at points far from the tip of the nose curve. At $\lambda = 0$, the circuit is under no-load and the initial voltage magnitude and angle are $1\angle 0^\circ$. With $\sigma = 0.1$ pu, the predictor step yields:

$$\begin{bmatrix} \delta \\ V \\ \lambda \end{bmatrix}^{\text{predicted}} = \begin{bmatrix} d\delta \\ dV \\ d\lambda \end{bmatrix} + \sigma \begin{bmatrix} & & \vdots & \\ & J_{LF} & \vdots & K \\ & & \vdots & \\ \cdots & \cdots & \cdots & \cdots \\ & & [e_k] & \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{3.170}$$

$$= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \sigma \begin{bmatrix} -0.9901 & -0.0988 & -1 \\ 0.0988 & -0.9901 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.171}$$

$$= \begin{bmatrix} -0.1000 \\ 0.9900 \\ 0.1000 \end{bmatrix} \tag{3.172}$$

where δ is in radians. Note that the predicted value for λ is 0.1 pu.

The corrector step solves the system of equations:

$$0 = -\lambda - 0.995V\cos(\delta - 95.7^\circ) - 0.995V^2\cos(84.3^\circ) \tag{3.173}$$

$$0 = -0.995V\sin(\delta - 95.7^\circ) - 0.995V^2\sin(84.3^\circ) \tag{3.174}$$

with the load parameter λ set to 0.1 pu. Note that this is a regular loadflow problem and can be solved without program modification.

The first corrector step yields

$$\begin{bmatrix} \delta \\ V \\ \lambda \end{bmatrix} = \begin{bmatrix} -0.1017 \\ 0.9847 \\ 0.1000 \end{bmatrix}$$

Note that this procedure is consistent with the illustration in Figure 3.15. The prediction step is of length σ taken tangentially to the PV at the current point. The corrector step will then occur along a vertical path because the power (λK) is held constant during the correction.

Iteration 2

The second iteration proceeds as the first. The predictor step yields the following guess:

$$\begin{bmatrix} \delta \\ V \\ \lambda \end{bmatrix} = \begin{bmatrix} -0.2060 \\ 0.9637 \\ 0.2000 \end{bmatrix}$$

where λ is increased by the stepsize σ = 0.1 pu.

Correcting the values yields the second update:

$$\begin{bmatrix} \delta \\ V \\ \lambda \end{bmatrix} = \begin{bmatrix} -0.2105 \\ 0.9570 \\ 0.2000 \end{bmatrix}$$

Iterations 3 and 4

The third and fourth iterations progress similarly. The values to this point are summarized:

λ	V	δ	σ
0.1000	0.9847	-0.1017	0.1000
0.2000	0.9570	-0.2105	0.1000
0.3000	0.9113	-0.3354	0.1000
0.4000	0.8268	-0.5050	0.1000

Beyond this point, the loadflow fails to converge for a stepsize of σ = 0.1. The method is nearing the point of maximum power flow (the tip of the nose curve) as indicated by the rapid decline in voltage for relatively small changes in λ. At this point, the continuation parameter is switched from λ to V to ensure that the corrector step will converge. The predictor step is modified such that:

$$\begin{bmatrix} d\delta \\ dV \\ d\lambda \end{bmatrix} = \begin{bmatrix} d\delta_0 \\ dV_0 \\ d\lambda_0 \end{bmatrix} + \sigma \begin{bmatrix} [J_{LF}] & \begin{matrix} -\lambda \\ 0 \end{matrix} \\ \begin{matrix} 0 & -1 \end{matrix} & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

where the -1 in the last row (the e_k vector) now corresponds to V rather than λ. The minus sign indicates that the predictor step will reduce the voltage magnitude by the stepsize σ. The stepsize σ is reduced to 0.025pu, which is a value more appropriate for changes in voltage magnitude.

The corrector step is also modified when the continuation parameter switches to voltage magnitude. The new augmented equations become:

$$0 = f_1(\delta, V, \lambda) = -\lambda - 0.995V\,(\cos(\delta - 95.7^\circ) + V\cos(84.3^\circ)) \quad (3.175)$$

$$0 = f_2(\delta, V, \lambda) = -0.995V\sin(\delta - 95.7^\circ) - 0.995V^2\sin(84.3^\circ) \quad (3.176)$$

$$0 = f_3(\delta, V, \lambda) = V - V^{\text{predicted}} \tag{3.177}$$

which cannot be solved with a traditional powerflow program due to the last equation. This equation is necessary to keep the Newton-Raphson iteration nonsingular. Fortunately, the Newton-Raphson iteration uses the same iteration matrix as the predictor matrix:

$$\begin{bmatrix} & & -\lambda \\ & [J_{LF}] & 0 \\ 0 & -1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \left(\begin{bmatrix} \delta \\ V \\ \lambda \end{bmatrix}^{(k+1)} - \begin{bmatrix} \delta \\ V \\ \lambda \end{bmatrix}^{(k)} \right) = - \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \tag{3.178}$$

thus minimizing the computational requirement.

Note that the corrector step is now a horizontal correction in voltage. The voltage magnitude is held constant while λ and δ are corrected. These iterates proceed as

Predicted				Corrected		
λ	V	δ	σ	λ	V	δ
0.4196	0.8019	-0.5487	0.0250	0.4174	0.8019	-0.5474
0.4326	0.7769	-0.5887	0.0250	0.4307	0.7769	-0.5876
0.4422	0.7519	-0.6268	0.0250	0.4405	0.7519	-0.6260
0.4487	0.7269	-0.6635	0.0250	0.4472	0.7269	-0.6627
0.4525	0.7019	-0.6987	0.0250	0.4511	0.7019	-0.6981
0.4538	0.6769	-0.7328	0.0250	0.4525	0.6769	-0.7323
0.4528	0.6519	-0.7659	0.0250	0.4517	0.6519	-0.7654

Note that at the last updates, the load parameter λ has started to decrease with decreasing voltage. This indicates that the continuation power flow is now starting to map out the lower half of the nose curve. However, while the iterations are still close to the tip of the nose curve, the Jacobian will still be ill conditioned, so it is a good idea to take several more steps before switching the continuation parameter from voltage magnitude back to λ.

Predicted				Corrected		
λ	V	δ	σ	λ	V	δ
0.4497	0.6269	-0.7981	0.0250	0.4487	0.6269	-0.7977
0.4447	0.6019	-0.8295	0.0250	0.4438	0.6019	-0.8291
0.4380	0.5769	-0.8602	0.0250	0.4371	0.5769	-0.8598
0.4296	0.5519	-0.8902	0.0250	0.4288	0.5519	-0.8899
0.4197	0.5269	-0.9197	0.0250	0.4190	0.5269	-0.9194

After switching the continuation parameter back to λ, the e_k vector becomes

$$e_k = [0 \quad 0 \quad -1]$$

where the -1 indicates that the continuation parameter λ will be decreasing (i.e., the power is decreasing back to the base case). The predictor/corrector steps proceed as above yielding

Predicted				Corrected		
λ	V	δ	σ	λ	V	δ
0.3190	0.2899	-1.1964	0.1000	0.3190	0.3564	-1.1088
0.2190	0.2187	-1.2554	0.1000	0.2190	0.2317	-1.2387
0.1190	0.1165	-1.3565	0.1000	0.1190	0.1220	-1.3496
0.0190	0.0166	-1.4553	0.1000	0.0190	0.0191	-1.4523

These values are combined in the PV curve shown in Figure 3.17. Note the change in step size when the continuation parameter switches from λ to voltage near the tip of the PV curve. The appropriate choice of step size is problem dependent and can be adaptively changed to improve computational e ciency.

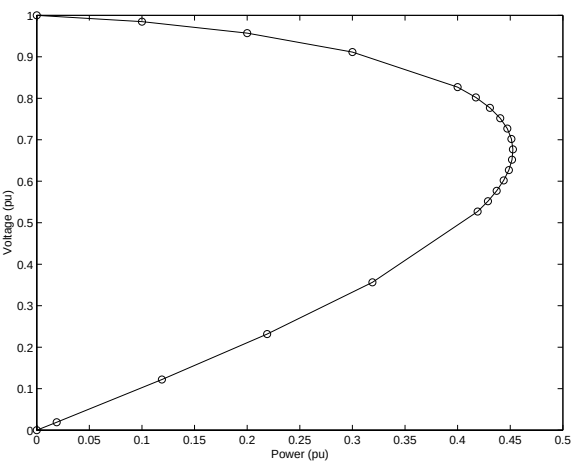


FIGURE 3.17
PV curve for system of Example 3.13

3.6.6 Three-Phase Power Flow

Another special purpose power flow application is for three-phase power flow analysis. Although much of the power system analysis is performed on balanced three phase systems using one line equivalent diagrams, certain situations call for a three phase analysis. In particular, in the situation where the transmission system is not balanced due to non-transposed lines or when the

loads are considerably unbalanced, it may be desirable to perform a full three phase load flow to ascertain the e ect on the individual line flows and bus voltages. The development of the three phase load flow is similar to that of the single phase equivalent, except that the mutual coupling between lines must be incorporated into the admittance matrix, yielding a $3n \times 3n$ matrix with elements Y_{ij}^{pq} where the subscript ij indicates bus number ($1 \leq (i,j) \leq n$) and the superscript pq indicates phase ($p,q \quad [a,b,c]$).

The incorporation of each phase individually leads to the similar, but slightly more complex, three-phase load flow equations:

$$0 = \Delta P_i^p = P_i^{inj,p} - V_i^p \sum_{q\in(a,b,c)} \sum_{j=1}^{N_{bus}} V_j^q Y_{ij}^{pq} \cos\left(\theta_i^p - \theta_j^q - \phi_{ij}^{pq}\right) \quad (3.179)$$

$$0 = \Delta Q_i^p = Q_i^{inj,p} - V_i^p \sum_{q\in(a,b,c)} \sum_{j=1}^{N_{bus}} V_j^q Y_{ij}^{pq} \sin\left(\theta_i^p - \theta_j^q - \phi_{ij}^{pq}\right) \quad (3.180)$$

$$i = 1,\ldots,N_{bus} \text{ and } p \quad (a,b,c)$$

There are three times as many load flow equations as in the single phase equivalent load flow equations. Generator (PV) buses are handled similarly with the following exceptions:

1. $\theta^a = 0°, \theta^b = -120°, \theta^c = 120°$ for the swing bus
2. All generator voltage magnitudes and active powers in each phase must be equal since generators are designed to produce balanced output

A three-phase load flow "flat start" is to set each bus voltage magnitude to

$$V_i^a = 1.0\angle 0°$$
$$V_i^b = 1.0\angle -120°$$
$$V_i^c = 1.0\angle 120°$$

The system Jacobian used in the Newton-Raphson solution of the load flow equations will have a possible $(3(2n) \times 3(2n))$ or $36n^2$ entries. The Jacobian partial derivatives are found in the same manner as with the single phase equivalent system except that the derivatives must also be taken with respect to phase di erences. For example,

$$\frac{\partial \Delta P_i^a}{\partial \theta_j^b} = V_i^a V_j^b Y_{ij}^{ab} \sin\left(\theta_i^a - \theta_j^b - \phi_{ij}^{ab}\right) \quad (3.181)$$

which is similar to the single phase equivalent system. Similarly

$$\frac{\partial \Delta P_i^a}{\partial \theta_i^a} = -V_i^a \sum_{q\in(a,b,c)} \sum_{j=1}^{N_{bus}} V_j^q Y_{ij}^{pq} \sin\left(\theta_i^p - \theta_j^q - \phi_{ij}^{pq}\right) + \left(V_i^a\right)^2 Y_{ii}^{pp} \cos\left(\phi_{ii}^{pp}\right) \quad (3.182)$$

The remaining partial derivatives can be calculated in a similar manner and the solution process of the three-phase power flow follows the method outlined in Section 3.6.1.

3.7 Problems

1. Prove that the Newton-Raphson iteration will diverge for the following functions regardless of choice of initial condition

   (a) $f(x) = x^2 + 1$

   (b) $f(x) = 7x^4 + 3x^2 + \pi$

2. Devise an iteration formula for computing the fifth root of any positive real number.

3. Using the Newton-Raphson method, solve

$$0 = 4y^2 + 4y + 52x - 19$$
$$0 = 169x^2 + 3y^2 + 111x - 10y - 10$$

   with $[x^0\ y^0]^T = [1\ 1]^T$.

4. Using the Newton-Raphson method, solve

$$0 = x - 2y + y^2 + y^3 - 4$$
$$0 = -xy + 2y^2 - 1$$

   with $[x^0\ y^0]^T = [1\ 1]^T$.

5. Repeat Problems 3 and 4 using numerical di erentiation to compute the Jacobian. Use a perturbation of 1% in each variable.

6. Repeat Problems 3 and 4 using the secant method.

7. Repeat Problems 3 and 4 using the homotopic mapping with $0 = f_{01} = x_1^2 - 2$ and $0 = f_{02} = x_2^2 - 4$

8. Write a *generalized* (for any system) power flow program. Your program should:

   (a) Read in load, voltage, and generation data. You may assume that bus #1 corresponds to the swing bus.

   (b) Read in line and transformer data and create the Y_{bus} matrix.

   (c) Solve the power flow equations using the Newton-Raphson algorithm, for a stopping criterion of $\left\| f\left(x^k\right) \right\| \leq \epsilon = 0.0005$.

   (d) Calculate all dependent unknowns, line flows, and line losses.

   The Newton-Raphson portion of the program should call the lufact and permute subroutines. Your program should give you the option of using either a “flat start” or “previous values” as an initial guess. The easiest way to accomplish this is to read and write to the same data file. Note that the first run must be a “flat start” case.

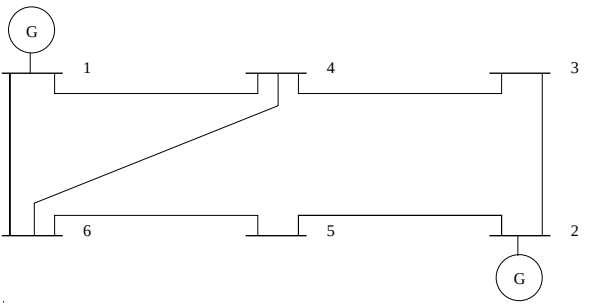


FIGURE 3.18
Ward-Hale 6 bus system

9. The data for the system shown in Figure 3.18 are given below:

No.	Type	$\|V\|$	θ	P_{gen}	Q_{gen}	P_{load}	Q_{load}
1	0	1.05	0	0	0	0.25	0.1
2	1	1.05	0	0.5	0	0.15	0.05
3	2	1.00	0	0	0	0.275	0.11
4	2	1.00	0	0	0	0	0
5	2	1.00	0	0	0	0.15	0.09
6	2	1.00	0	0	0	0.25	0.15

No.	To	From	R	X	B
1	1	4	0.020	0.185	0.009
2	1	6	0.031	0.259	0.010
3	2	3	0.006	0.025	0.000
4	2	5	0.071	0.320	0.015
5	4	6	0.024	0.204	0.010
6	3	4	0.075	0.067	0.000
7	5	6	0.025	0.150	0.017

Calculate the load flow solution for the system data given above. Remember to calculate all dependent unknowns, line-flows, and line losses.

10. Modify your loadflow program so that you are using a *decoupled load flow* (i.e., assume $\left[\frac{\partial \Delta P}{\partial V}\right] = 0$ and $\left[\frac{\partial \Delta Q}{\partial \theta}\right] = 0$). Repeat problem 9. Discuss the di erence in convergence between the decoupled and the full Newton-Raphson Power Flows.

11. Increase the line resistances by 75% (i.e. multiply all resistances by 1.75) and repeat problem 9 and problem 10. Discuss your findings.

12. Using a continuation power flow, map out the "PV" curve for the original system data by increasing/decreasing the load on bus 6 holding a constant P/Q ratio from $P = 0$ to the point of maximum power transfer.

13. Making the following assumptions, find a constant, decoupled Jacobian that could be used in a fast, decoupled 3-phase load flow.

    - $V_i^p \approx 1.0pu$ for all i and p
    - $\theta_{ij}^{pp} \approx 0$
    - $\theta_{ij}^{pm} \approx \pm 120^o \quad p = m$
    - $g_{ij}^{pm} << b_{ij}^{pm}$

4

Sparse Matrix Solution Techniques

A sparse matrix is one that has "very few" non-zero elements. A sparse system is one in which its mathematical description gives rise to sparse matrices. Any large system that can be described by coupled nodes may give rise to a sparse matrix if the majority of nodes in the system have very few connections. Many systems in engineering and science result in sparse matrix descriptions. Large systems in which each node is connected to only a handful of other nodes include the mesh points in a finite-element-analysis, nodes in an electronic circuit, and the busbars in an electric power network. For example, power networks may contain thousands of nodes (busbars), but the average connectivity of electric power network nodes is three; each node is connected, on average, to three other nodes. This means that in a system comprised of a thousand nodes, the non-zero percentage of the descriptive system matrix is

$$\frac{4 \text{ non-zeros elements}}{\text{row}} \times 1000 \text{ rows} \Big/ 1000 \times 1000 \text{ elements} \times 100\% = 0.4\% \text{ non-zero elements}$$

Thus, if only the non-zero elements were stored in memory, they would require only 0.4% of the memory requirements of the full 1000 × 1000 matrix. Full storage of an $n \times n$ system matrix grows as n^2, whereas the sparse storage of the same system matrix increases only linearly as n. Thus significant storage and computational savings can be realized by exploiting sparse storage and solution techniques. Another motivating factor in exploiting sparse matrix solution techniques is the computational e ort involved in solving matrices with large percentages of zero elements. Consider the solution of the linear problem

$$Ax = b$$

where A is sparse. The factorization of L and U from A requires a significant number of multiplications where one or both of the factors may be zero. If it is known ahead of time where the zero elements reside in the matrix, these multiplications can be avoided (since their product will be zero) and significant computational e ort can be saved. The salient point here is that these computations are skipped altogether. A person performing an LU factorization by hand can note which values are zero and skip those particular multiplications. A computer, however, does not have the ability to "see" the zero elements. Therefore the sparse solution technique must be formulated in such a way as to avoid zero computations altogether and operate only upon non-zero elements.

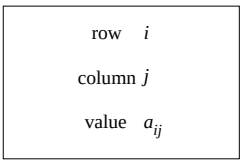


FIGURE 4.1
Basic storage element for a_{ij}

In this chapter, both the storage and computational aspects of sparse matrix solutions will be explored. Several storage techniques will be discussed and ordering techniques to minimize computational e ort will be developed.

4.1 Storage Methods

In sparse storage methods, only the non-zero elements of the $n \times n$ matrix A are stored in memory, along with the indexing information needed to transverse the matrix from element to element. Thus each element must be stored with its real value (a_{ij}) and its position indices (row and column) in the matrix. The basic storage unit may be visualized as the object shown in Figure 4.1.

In addition to the basic information, indexing information must also be included in the object, such as a link to the next element in the row, or the next element in the column, as shown in Figure 4.2.

The only additional information necessary to fully transverse the matrix either by row or column is an indication of the first element in each row or column. This is a stand-alone set of links that point to the first element in each row or column.

Example 4.1
Determine a linked list representation for the sparse matrix:

$$A = \begin{bmatrix} -1 & 0 & -2 & 0 & 0 \\ 2 & 8 & 0 & 1 & 0 \\ 0 & 0 & 3 & 0 & -2 \\ 0 & -3 & 2 & 0 & 0 \\ 1 & 2 & 0 & 0 & -4 \end{bmatrix}$$

Solution 4.1 A linked list representation of this matrix is shown in Figure 4.3. The last element of each column and row are linked to a null point. Note that each object is linked to its adjacent neighbors in the matrix, both

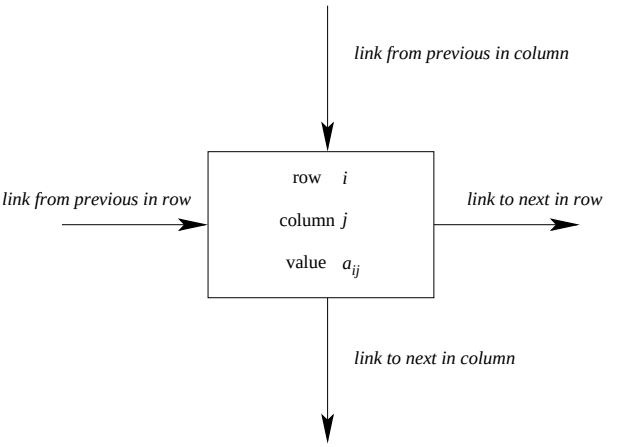


FIGURE 4.2
Storage element for element a_{ij} with links

by column and by row. In this way, the entire matrix can be transversed in any direction by starting at the first element and following the links until the desired element is reached.

If a command requires a particular matrix element, then by either choosing the column or row, the element can be located by progressing along the links. If during the search, the null point is reached, then the desired element does not exist and a value of zero is returned. Additionally, if the matrix elements are linked by increasing index and an element is reached that has a greater index than the desired element, then the progression terminates and a value of zero is returned.

A linked list representation for a sparse matrix is not unique and the elements do not necessarily have to be linked in increasing order by index. However, ordering the elements by increasing index leads to a simplified search since the progression can be terminated before reaching the null point if the index of the linked object exceeds the desired element. If the elements are not linked in order, the entire row or column must be transversed to determine whether or not the desired element is non-zero. The drawback to an ordered list is that the addition of new non-zero elements to the matrix requires the update of both row and column links.

Example 4.2
Insert the matrix element $A(4,5) = 10$ to the linked list of Example 4.1.

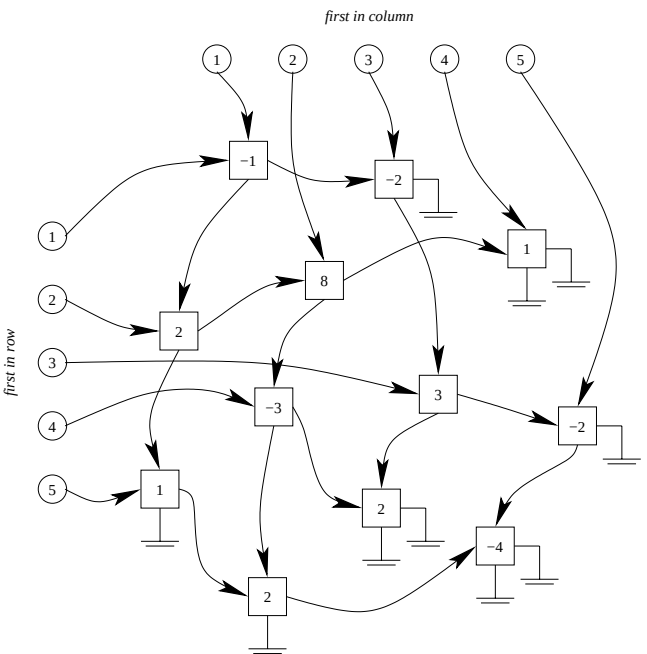


FIGURE 4.3
Linked list for Example 4.1

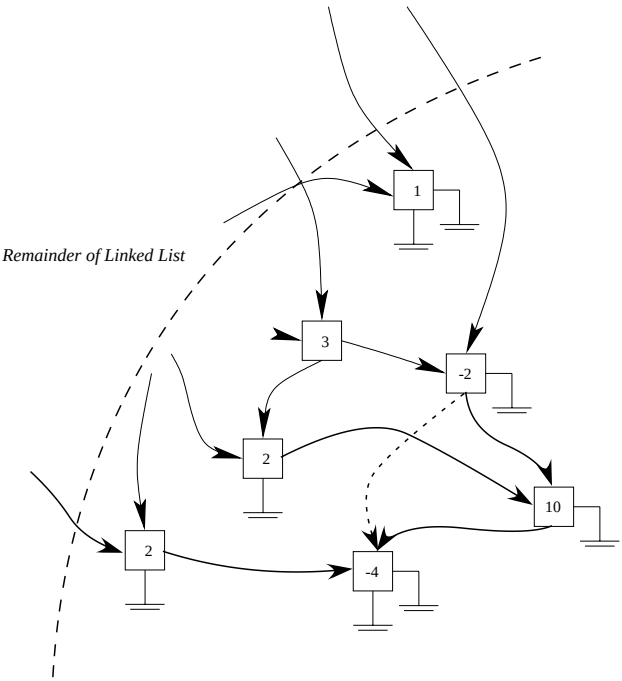


FIGURE 4.4
Insertion of matrix element $A(4,5) = 10$

Solution 4.2 The insertion of the new element is shown in Figure 4.4. The addition of this element requires two transversals of the matrix to insert the new element and to update the links; one transversal by row and one by column. Starting at the *first in row* link for row 4 (value$=-3$), the elements are progressed by link. Following the links and monitoring the column indices, it is discovered that the element with column index 3 (value$=2$) is the last element in the row since it points to null. Since the new element has column index 5, it is inserted between the (value$=2$) element and the null point in the row linked list. Similarly, starting at the *first in column* link for column 5 (value$=-2$), the column is transversed and inserted between the elements with row indices 3 (value$=-2$) and 5 (value$=-4$). The column links are updated to reflect the insertion.

If the linked lists of the matrix are not ordered by index, then new elements can be added without transversing the rows or columns. A new element can be inserted in each row or column by inserting them before the first element and updating the *first in row* and *first in column* pointers.

Many software languages, however, do not support the use of objects, pointers, and linked lists. In this case it is necessary to develop a procedure to mimic a linked list format by the use of vectors. Three vectors are required to represent each non-zero element object: one vector containing the row number (NROW), one vector containing the column number (NCOL), and one vector containing the value of the element (VALUE). These vectors are of length nnz where nnz is the number of non-zero elements. Two vectors, also of length nnz, are required to represent the next-in-row links (NIR) and the next-in-column (NIC) links. If an element is the last in the row or column, then the NIR or NIC value for that element is 0. Lastly, two vectors of length n contain the *first in row* (FIR) and *first in column* (FIC) links.

The elements of the matrix are assigned a (possibly arbitrary) numbering scheme that corresponds to their order in the NROW, NCOL, VALUE, NIR, and NIC vectors. This order is the same for each of these five vectors. The FIR and FIC vectors will also refer to this number scheme.

Example 4.3

Find the vectors NROW, NCOL, VALUE, NIR, NIC, FIR, and FIC for the sparse matrix of Example 4.1.

Solution 4.3 The matrix of Example 4.1 is reproduced below with the numbering scheme given in parentheses to the left of each non-zero element. The numbering scheme is sequential by row and goes from 1 to nnz = 12.

$$A = \begin{bmatrix} (1)\ -1 & 0 & (2)\ -2 & 0 & 0 \\ (3)\ 2 & (4)\ 8 & 0 & (5)\ 1 & 0 \\ 0 & 0 & (6)\ 3 & 0 & (7)\ -2 \\ 0 & (8)\ -3 & (9)\ 2 & 0 & 0 \\ (10)\ 1 & (11)\ 2 & 0 & 0 & (12)\ -4 \end{bmatrix}$$

The ordering scheme indicated yields the following nnz vectors:

k	VALUE	NROW	NCOL	NIR	NIC
1	-1	1	1	2	3
2	-2	1	3	0	6
3	2	2	1	4	10
4	8	2	2	5	8
5	1	2	4	0	0
6	3	3	3	7	9
7	-2	3	5	0	12
8	-3	4	2	9	11
9	2	4	3	0	0
10	1	5	1	11	0
11	2	5	2	12	0
12	-4	5	5	0	0

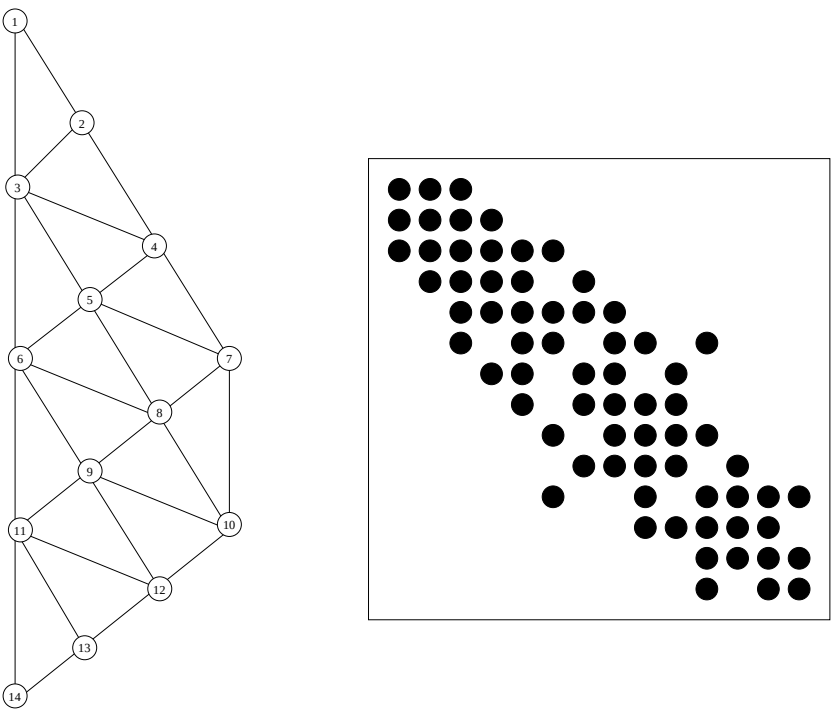
and the following n vectors:

	FIR	FIC
1	1	1
2	3	4
3	6	2
4	8	5
5	10	7

Consider the matrix element $A(2,2) = 8$. It is the 4th element in the numbering scheme, so its information is stored in the fourth place in vectors VALUE, NROW, NCOL, NIR, and NIC. Thus VALUE(4)=8, NROW(4)=2, and NCOL(4)=2. The next element in row 2 is $A(2,4) = 1$ and it is element 5 in the numbering scheme. Therefore NROW(4)=5, signifying that element 5 follows element 4 in its row (note however that it does not indicate which row they are in). Similarly, the next element in column 2 is $A(4,2) = -3$ and it is element 8 in the numbering scheme. Therefore NCOL(4)=8.

4.2 Sparse Matrix Representation

Sparse matrices arise as the result of the mathematical modeling of a sparse system. In many cases, the system has a naturally occurring physical network representation or lends itself to a physically intuitive representation. In these cases, it is often informative to visualize the connectivity of the system by graphical means. In the graphical representation, each node of the graph corresponds to a node in the system. Each edge of the graph corresponds to a branch of the network. As with a network, the graph, consisting of vertices and edges, is often represented by a set of points in the plane joined by a line representing each edge. Matrices that arise from the mathematical model of a graphically represented network are structurally symmetric. In other words, if the matrix element a_{ij} is non-zero, then the matrix element a_{ji} is



(a) (b)

FIGURE 4.5
(a) A finite element grid model, (b) The corresponding matrix

also non-zero. This implies that if node i is connected to node j, then node j is also connected to node i. Matrices that are not structurally symmetric can be made symmetric by adding an element of value zero in the appropriate position within the matrix.

In addition to a graphical representation, it is also common to visualize sparse matrices by a matrix that is clear except for an identifying symbol (such as a $\times, \bullet,$, or other mark) to represent the position of the non-zero elements in the matrix. The finite element grid of the trapezoid shown in Figure 4.5(a) gives rise to the sparse matrix structure shown in Figure 4.5(b). Note that the ordering of the matrix is not unique; another numbering scheme for the nodes will result in an alternate matrix structure.

4.3 Ordering Schemes

Node ordering schemes are important in minimizing the number of multiplications and divisions required for both L and U triangularization and forward/backward substitution. A good ordering will result in the addition of few *fills* to the triangular factors during the LU factorization process. A *fill* is a non-zero element in the L or U matrix that was zero in the original A matrix. If A is a full matrix, $\alpha = \frac{n^3-n}{3}$ multiplications and divisions are required for the LU factorization process and $\beta = n^2$ multiplications and divisions are required for the forward/backward substitution process. The number of multiplications and divisions required can be substantially reduced in sparse matrix solutions if a proper node ordering is used.

Example 4.4

Determine the number of multiplications, divisions, and fills required for the solution of the system shown in Figure 4.6.

Solution 4.4 The LU factorization steps yield

$$
\begin{aligned}
q_{11} &= a_{11} \\
q_{21} &= a_{21} \\
q_{31} &= a_{31} \\
q_{41} &= a_{41} \\
q_{51} &= a_{51}
\end{aligned}
$$

$$
\begin{aligned}
q_{12} &= a_{12}/q_{11} \\
q_{13} &= a_{13}/q_{11}
\end{aligned}
$$

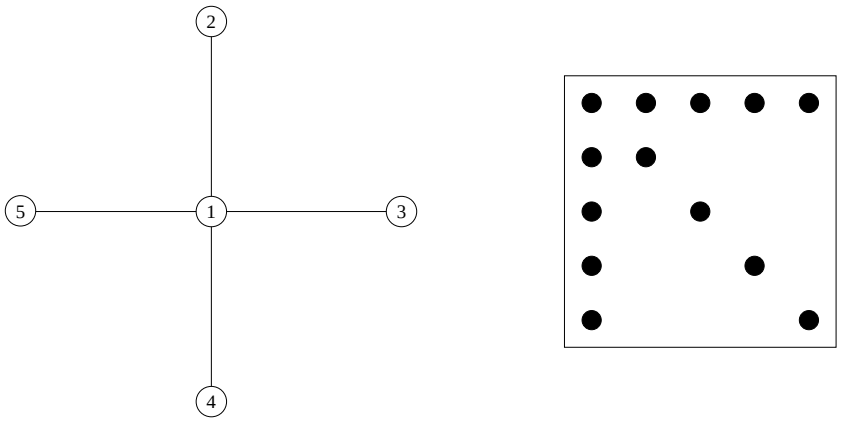


FIGURE 4.6
Graph and matrix for Example 4.4

$$q_{14} = a_{14}/q_{11}$$
$$q_{15} = a_{15}/q_{11}$$

$$q_{22} = a_{22} - q_{21}q_{12}$$
$$q_{32} = a_{32} - q_{31}q_{12}$$
$$q_{42} = a_{42} - q_{41}q_{12}$$
$$q_{52} = a_{52} - q_{51}q_{12}$$

$$q_{23} = (a_{23} - q_{21}q_{13})/q_{22}$$
$$q_{24} = (a_{24} - q_{21}q_{14})/q_{22}$$
$$q_{25} = (a_{25} - q_{21}q_{15})/q_{22}$$

$$q_{33} = a_{33} - q_{31}q_{13} - q_{32}q_{23}$$
$$q_{43} = a_{43} - q_{41}q_{13} - q_{42}q_{23}$$
$$q_{53} = a_{53} - q_{51}q_{13} - q_{52}q_{23}$$

$$q_{34} = (a_{34} - q_{31}q_{14} - q_{32}q_{24})/q_{33}$$
$$q_{35} = (a_{35} - q_{31}q_{15} - q_{32}q_{25})/q_{33}$$

$$q_{44} = a_{44} - q_{41}q_{14} - q_{42}q_{24} - q_{43}q_{34}$$
$$q_{54} = a_{54} - q_{51}q_{14} - q_{52}q_{24} - q_{53}q_{34}$$

$$q_{45} = (a_{45} - q_{41}q_{15} - q_{22}q_{25} - q_{43}q_{35})/q_{44}$$

$$q_{55} = a_{55} - q_{51}q_{15} - q_{52}q_{25} - q_{53}q_{35} - q_{54}q_{45}$$

The multiplications and divisions required for the LU factorization are summarized by row and column.

row	column	multiplications	divisions	fills
1		0	0	
	1	0	4	
2		4	0	a_{32}, a_{42}, a_{52}
	2	3	3	a_{23}, a_{24}, a_{25}
3		6	0	a_{43}, a_{53}
	3	4	2	a_{34}, a_{35}
4		6	0	a_{54}
	4	3	1	a_{45}
5		4	0	

Therefore $\alpha = 40$ is the total number of multiplications and divisions in the LU factorization. The forward ($Ly = b$) and backward ($Ux = y$) substitution steps yield:

$$\begin{aligned}
y_1 &= b_1/q_{11} \\
y_2 &= (b_2 - q_{21}y_1)/q_{22} \\
y_3 &= (b_3 - q_{31}y_1 - q_{32}y_2)/q_{33} \\
y_4 &= (b_4 - q_{41}y_1 - q_{42}y_2 - q_{43}y_3)/q_{44} \\
y_5 &= (b_5 - q_{51}y_1 - q_{52}y_2 - q_{53}y_3 - q_{54}y_4)/q_{55} \\
x_5 &= y_5 \\
x_4 &= y_4 - q_{45}x_5 \\
x_3 &= y_3 - q_{35}x_5 - q_{34}x_4 \\
x_2 &= y_2 - q_{25}x_5 - q_{24}x_4 - q_{23}x_3 \\
x_1 &= y_1 - q_{15}x_5 - q_{14}x_4 - q_{13}x_3 - q_{12}x_2
\end{aligned}$$

	forward		backward	
row	multiplications	divisions	multiplications	divisions
1	0	1	4	0
2	1	1	3	0
3	2	1	2	0
4	3	1	1	0
5	4	1	0	0

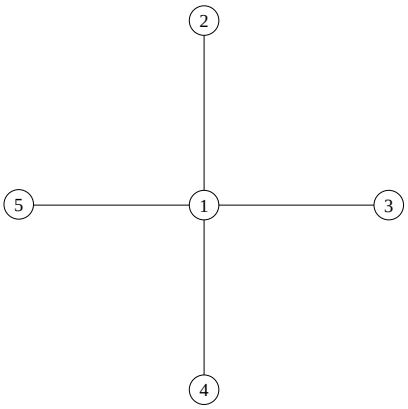


FIGURE 4.7
Graph for Example 4.4

Thus $\beta = 25$ is the total number of multiplications and divisions in the forward and backward substitution steps. The total number of multiplications and divisions for the solution of $Ax = b$ is $\alpha + \beta = 36$.

A fill occurs when a matrix element that was originally zero becomes non-zero during the factorization process. This can be visually simulated using a graphical approach. Consider the graph of Example 4.4 shown again in Figure 4.7.

In this numbering scheme, the row and column corresponding to node 1 is factorized first. This corresponds to the removal of node 1 from the graph. When node 1 is removed, all of the vertices to which it was connected must then be joined. Each edge added represents two fills in the Q matrix (q_{ij} and q_{ji}) since Q is symmetric. The graph after the removal of node 1 is shown in Figure 4.8. The dashed lines indicate that six fills will occur as a result: q_{23}, q_{24}, q_{25}, q_{34}, q_{35}, and q_{45}. These are the six fills that are also listed in the solution of the example.

Example 4.5

Determine the number of multiplications, divisions, and fills required for the solution of the system shown in Figure 4.9.

Solution 4.4 The LU factorization steps yield

$$q_{11} = a_{11}$$
$$q_{51} = a_{51}$$
$$q_{15} = a_{15}/q_{11}$$

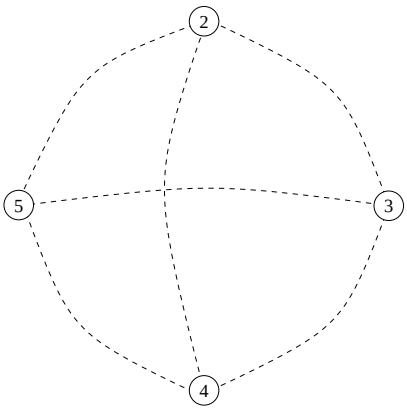


FIGURE 4.8
Resulting fills after removing node 1

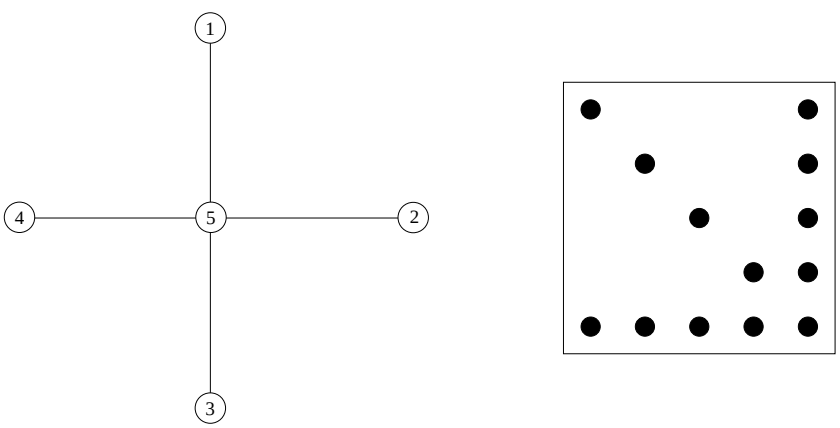


FIGURE 4.9
Graph and matrix for Example 4.5

$$
\begin{aligned}
q_{22} &= a_{22} \\
q_{25} &= a_{25}/q_{22} \\
q_{52} &= a_{52} \\
q_{33} &= a_{33} \\
q_{53} &= a_{53} \\
q_{35} &= a_{35}/q_{33}
\end{aligned}
$$

$$
\begin{aligned}
q_{44} &= a_{44} \\
q_{54} &= a_{54} \\
q_{45} &= a_{45}/q_{44} \\
q_{55} &= a_{55} - q_{51}q_{15} - q_{52}q_{25} - q_{53}q_{35} - q_{54}q_{45}
\end{aligned}
$$

The multiplications and divisions required for the LU factorization are summarized by row and column.

row	column	multiplications	divisions	fills
1		0	0	
	1	0	1	
2		0	0	
	2	0	1	
3		0	0	
	3	0	1	
4		0	0	
	4	0	1	
5		4	0	

Therefore $\alpha = 8$ is the total number of multiplications and divisions in the LU factorization. The forward ($Ly = b$) and backward ($Ux = y$) substitution steps yield:

$$
\begin{aligned}
y_1 &= b_1/q_{11} \\
y_2 &= b_2/q_{22} \\
y_3 &= b_3/q_{33} \\
y_4 &= b_4/q_{44} \\
y_5 &= (b_5 - q_{51}y_1 - q_{52}y_2 - q_{53}y_3 - q_{54}y_4)/q_{55} \\
x_5 &= y_5 \\
x_4 &= y_4 - q_{45}x_5 \\
x_3 &= y_3 - q_{35}x_5 \\
x_2 &= y_2 - q_{25}x_5 \\
x_1 &= y_1 - q_{15}x_5
\end{aligned}
$$

	forward		backward	
row	multiplications	divisions	multiplications	divisions
1	0	1	1	0
2	0	1	1	0
3	0	1	1	0
4	0	1	1	0
5	4	1	0	0

Thus $\beta = 13$ is the total number of multiplications and divisions in the forward and backward substitution steps. The total number of multiplications and divisions for the solution of $Ax = b$ is $\alpha + \beta = 21$.

Even though both original matrices had the same number of non-zero elements, there is a significant reduction in the number of multiplications and divisions by simply renumbering the vertices of the matrix graph. This is due, in part, by the number of fills that occurred during the LU factorization of the matrix. The Q matrix of Example 4.4 became full, whereas the Q matrix of Example 4.5 retained the same sparse structure as the original A matrix. From these two examples, it can be concluded that although various node orders do not a ect the accuracy of the linear solution, the ordering scheme greatly a ects the time in which the solution is achieved. A good ordering scheme is one in which the resulting Q matrix has a similar structure to the original A matrix. This means that the number of fills is minimized. This objective forms the basis for a variety of ordering schemes. The problem of optimal ordering is an NP-complete problem [54], but several schemes have been developed that provide near-optimal results.

Example 4.6

Determine number of fills, α, and β for the matrix shown in Figure 4.10 as currently ordered.

Solution 4.6 The first step is to determine where the fills from LU factorization will occur. By observation, the fills will occur in the places designated by the in the matrix shown in Figure 4.11. From the figure, the number of fills is 24.

Rather than calculating the number of multiplications and divisions required for LU factorization and forward/backward substitution, there is a handy way of calculating α and β directly from the filled matrix.

$$\alpha = \sum_{i=1}^{n} (\text{nnz in column } i \text{ below } q_{ii} + 1) \times (\text{nnz in row } i \text{ to right of } q_{ii}) \quad (4.1)$$

$$\beta = nnz \text{ of matrix } Q \quad (4.2)$$

Using equations (4.1) and (4.2),

$$\alpha = (3 \times 4) + (4 \times 5) + (5 \times 6) + (4 \times 5) + (4 \times 5) + (3 \times 4) + (3 \times 4) + (2 \times 3) + (1 \times 2) + (0 \times 1) = 134$$

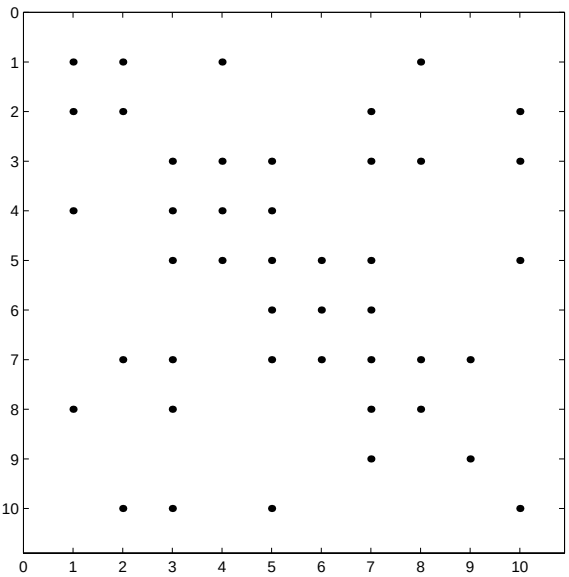


FIGURE 4.10
Matrix for Example 4.6

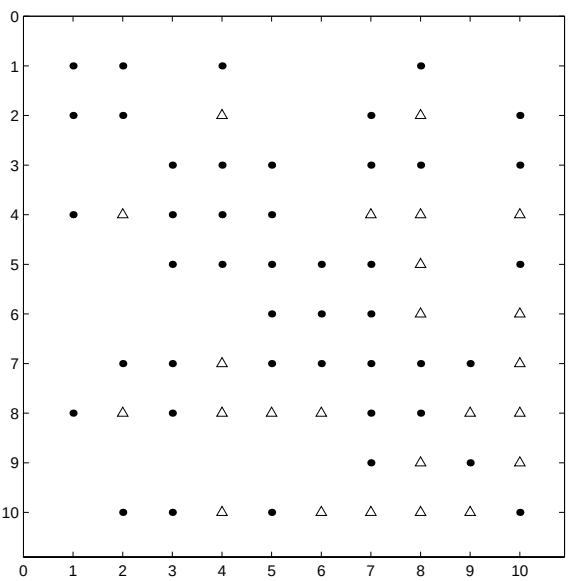


FIGURE 4.11
Matrix with fills for Example 4.6

and $\beta = nnz = 68$ for the Q matrix shown in Figure 4.11, thus $\alpha + \beta = 202$. Compare this with the $\alpha + \beta = 430$.

Even without an ordering scheme, the sparse matrix solution process yields over a 50% reduction in computation. One goal of an ordering scheme is to introduce the least number of fills in the factored matrix Q to minimize the number of multiplications and divisions α. A second goal is also to minimize β, which is the number of multiplications and divisions in the forward/backward substitution step. These dual objectives lead to several approaches to ordering.

4.3.1 Scheme 0

From Examples 4.4 and 4.5, it can be generalized that a better ordering is achieved if the nodes are ordered into a lower-right pointing "arrow" matrix. One rapid method of achieving this e ect is to number the nodes according to their degree, where the degree of a node is defined as the number of edges connected to it. In this scheme, the nodes are ordered from lowest degree to highest degree.

Scheme 0

1. Calculate the degree of all vertices.
2. Choose the node with the lowest degree. Place in ordering scheme.
3. In case of a tie, choose node with lowest natural ordering.
4. Return to step 2.

Example 4.7

Using Scheme 0, reorder the matrix of Example 4.6. Calculate α, β and the number of fills for this ordering.

Solution 4.7 The degrees of each of the nodes are given below:

node	degree
1	3
2	3
3	5
4	3
5	5
6	2
7	6
8	3
9	1
10	3

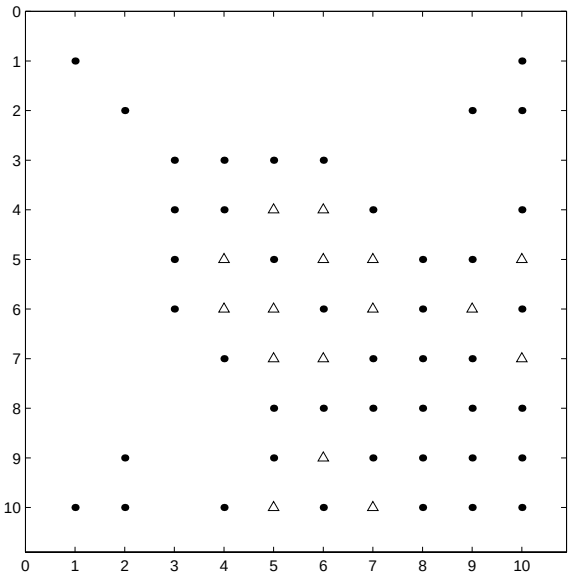


FIGURE 4.12
Matrix with fills for Example 4.7

Applying Scheme 0, the new ordering is

$$\text{Ordering } 0 = [9\ \ 6\ \ 1\ \ 2\ \ 4\ \ 8\ \ 10\ \ 3\ \ 5\ \ 7]$$

Reordering the matrix of Example 4.6 to reflect this ordering yields the matrix (with fills) shown in Figure 4.12. Note how the non-zeros elements begin to resemble the desired lower-right pointing arrow. The Scheme 0 ordering results in 16 fills as compared to 24 with the original ordering. From the matrix and equations (4.1) and (4.2), $\alpha = 110$ and $\beta = 60$, thus $\alpha + \beta = 170$ which is a considerable reduction over the original $\alpha + \beta = 202$.

4.3.2 Scheme I

Scheme 0 o ers simplicity and speed of generation, but does not directly take into account the e ect of fills on the ordering procedure. To do this, the e ect of eliminating the nodes as they are ordered must be taken into account. This modification is given in Scheme I.

Scheme I

1. Calculate the degree of all vertices.
2. Choose the node with the lowest degree. Place in ordering scheme. Eliminate it and update degrees accordingly.

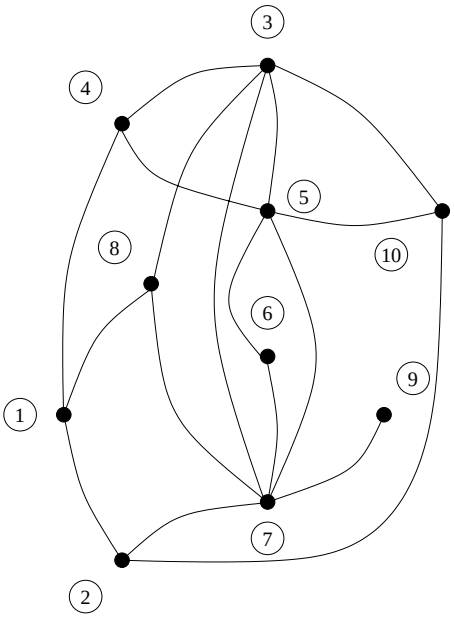


FIGURE 4.13
Graph of the matrix in Figure 4.10

3. In case of a tie, choose node with lowest natural ordering.

4. Return to step 1.

Scheme I is also known by many names including the Markowitz algorithm [31], the Tinney I algorithm [50], or most generally as the minimum degree algorithm.

Example 4.8

Using Scheme I, reorder the matrix of Example 4.6. Calculate α, β and the number of fills for this ordering.

Solution 4.8 The ordering for Scheme I takes into account the e ect of fills on the ordering as nodes are placed in the ordering scheme and eliminated. This algorithm is best visualized using the graphical representation of the matrix. The graph of the orginal unordered matrix of Figure 4.10 is shown in Figure 4.13.

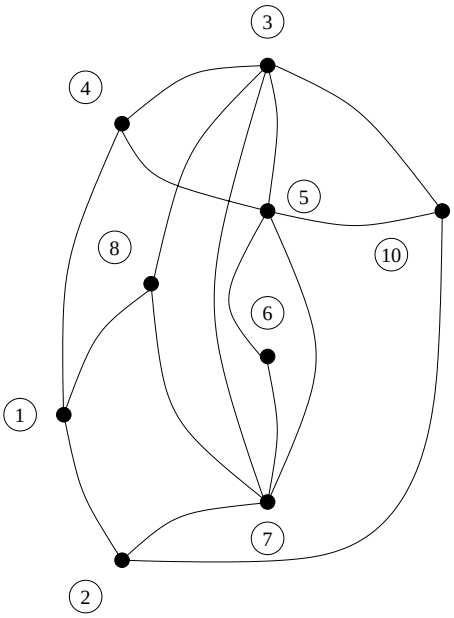


FIGURE 4.14
Updated graph with the removal of node 9

The degrees of each of the nodes are given below:

node	degree
1	3
2	3
3	5
4	3
5	5
6	2
7	6
8	3
9	1
10	3

From the degrees, the node with the lowest degree is ordered first. Node 9 has the lowest degree with only one connection. Its elimination does not cause any fills. The updated graph is shown in Figure 4.14.

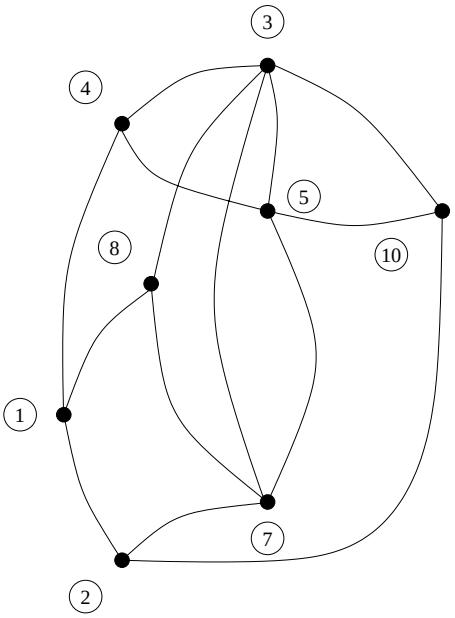


FIGURE 4.15
Updated graph with the removal of node 6

The updated degree of each of the nodes is given below:

node	degree
1	3
2	3
3	5
4	3
5	5
6	2
7	5
8	3
10	3

Node 7 now has one less degree. Applying the Scheme I algorithm again indicates that the next node to be chosen is node 6, with a degree of 2. Node 6 is connected to both node 5 and node 7. Since there is a pre-existing connection between these nodes, the elimination of node 6 does not create a fill between nodes 5 and 6. The elimination of node 6 is shown in Figure 4.15.

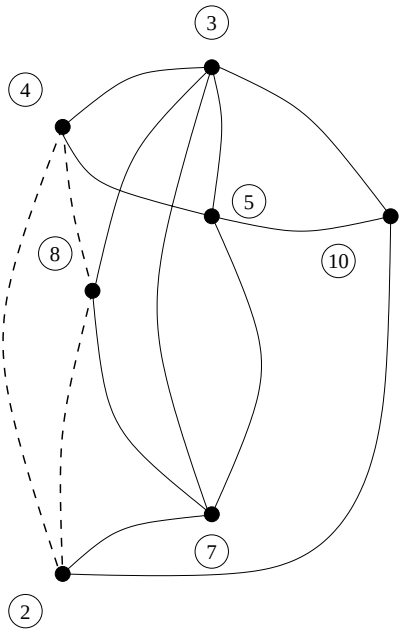


FIGURE 4.16
Updated graph with the removal of node 1

The new node degrees are

node	degree
1	3
2	3
3	5
4	3
5	4
7	4
8	3
10	3

As a result of the elimination of node 6, the degrees of nodes 5 and 7 decrease by one. Applying the Scheme I algorithm again indicates that the nodes with the lowest degrees are nodes [1 2 4 8 10]. Since there is a tie between these nodes, the node with the lowest natural ordering, node 1, is chosen and eliminated. Node 1 is connected to nodes 2, 4, and 8. None of these nodes is connected; therefore, the elimination of node 1 creates three fills: 4–8, 4–2, and 2–8. These fills are shown with the dashed edges in Figure 4.16.

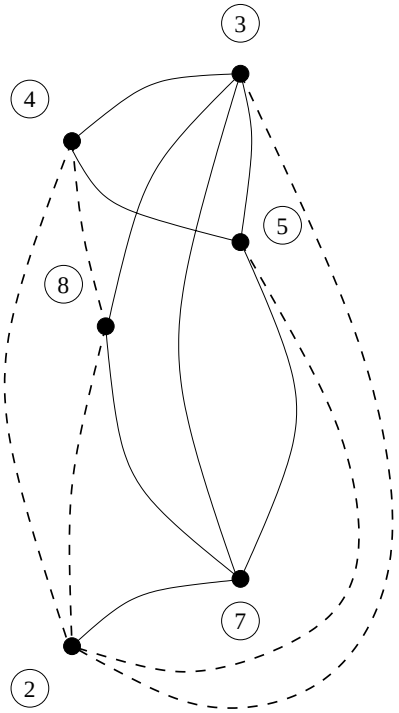


FIGURE 4.17
Updated graph with the removal of node 10

The new node degrees after the removal of node 1 are

node	degree
2	4
3	5
4	4
5	5
7	5
8	4
10	3

The addition of the three fills increased the degrees of nodes 2, 4, and 8. Applying the Scheme I algorithm again indicates that the node with the lowest degree is node 10. There is no tie in degree this time. Node 10 is chosen and eliminated. The elimination of node 10 creates two fills between nodes 2–5 and 2–3. These fills are shown with the dashed edges in Figure 4.17.

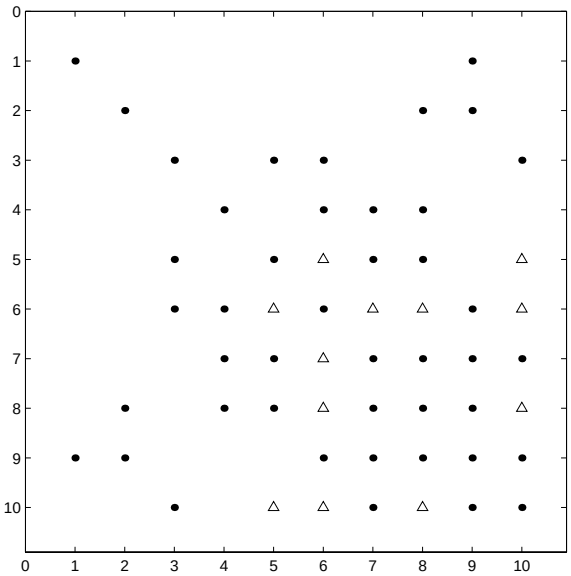


FIGURE 4.18
Matrix with fills for Example 4.8

Continuing to apply the Scheme I algorithm successively until all nodes have been chosen and eliminated yields the following final ordering:

$$\text{Ordering I} = [9\ \ 6\ \ 1\ \ 10\ \ 4\ \ 2\ \ 3\ \ 5\ \ 7\ \ 8]$$

Reordering the matrix of Example 4.8 to reflect this ordering yields the matrix (with fills) shown in Figure 4.18. Note how the non-zero elements continue to resemble the desired lower-right pointing arrow. The Scheme I ordering results in 12 fills as compared to 24 with the original ordering, and 16 with Scheme 0. From the matrix and equations (4.1) and (4.2), $\alpha = 92$ and $\beta = 56$, thus $\alpha + \beta = 148$ which is a considerable reduction over the original $\alpha + \beta = 202$, and the $\alpha + \beta = 170$ of Scheme 0.

4.3.3 Scheme II

Scheme 0 o ers a rapid way to order the nodes to give a quick "once-over" and obtain a reasonable ordering. It requires little computation beyond calculating the degrees of each node of the matrix. Scheme I takes this approach one step further. It still relies on the minimum-degree approach, but it includes a simulation of the LU factorization process to update the node degrees at each step of the factorization. One further improvement to this approach is

to develop a scheme that endeavors to minimize the number of fills at each step of the factorization. Thus, at each step, each elimination alternative is considered and the number of resulting fills is calculated. This scheme is also known as the Berry algorithm and the Tinney II algorithm and is summarized below:

Scheme II

1. For each node, calculate the number of fills that would result from its elimination.

2. Choose the node with the lowest number of fills.

3. In case of a tie, choose node with lowest degree.

4. In case of a tie, choose node with lowest natural ordering.

5. Place node in ordering scheme. Eliminate it and update fills and degrees accordingly.

6. Return to step 1.

Example 4.9

Using Scheme II, reorder the matrix of Example 4.6. Calculate α, β and the number of fills for this ordering.

Solution 4.9 The ordering for Scheme II takes into account the e ect of fills on the ordering as nodes are placed in the ordering scheme and eliminated. The degrees and resulting fills are given below:

node	degree	fills if eliminated	edges created
1	3	3	2–4, 2–8, 4–8
2	3	3	1–7, 1–10, 7–10
3	5	6	4–7, 4–8, 4–10, 5–8, 7–10, 8–10
4	3	2	1–3, 1–5
5	5	6	3–6, 4–6, 4–7, 4–10, 6–10, 7–10
6	2	0	none
7	6	12	2–3, 2–5, 2–6, 2–8, 2–9, 3–6, 3–9, 5–8, 5–9, 6–8, 6–9, 8–9
8	3	2	1–3, 1–7
9	1	0	none
10	3	2	2–3, 2–5

From this list, the elimination of nodes 6 or 9 will not result in any additional edges, or fills. Since there is a tie, the node with the lowest degree is chosen.

Thus, node 9 is chosen and eliminated. The number of fills and degrees is updated to apply the Scheme II algorithm again.

node	degree	fills if eliminated	edges created
1	3	3	2–4, 2–8, 4–8
2	3	3	1–7, 1–10, 7–10
3	5	6	4–7, 4–8, 4–10, 5–8, 7–10, 8–10
4	3	2	1–3, 1–5
5	5	6	3–6, 4–6, 4–7, 4–10, 6–10, 7–10
6	2	0	none
7	5	7	2–3, 2–5, 2–6, 2–8, 3–6, 5–8, 6–8
8	3	2	1–3, 1–7
10	3	2	2–3, 2–5

The next node to be eliminated is node 6 because it creates the fewest fills if eliminated. This node is therefore chosen and eliminated. The number of fills and degrees is again updated.

node	degree	fills if eliminated	edges created
1	3	3	2–4, 2–8, 4–8
2	3	3	1–7, 1–10, 7–10
3	5	6	4–7, 4–8, 4–10, 5–8, 7–10, 8–10
4	3	2	1–3, 1–5
5	5	6	3–6, 4–6, 4–7, 4–10, 6–10, 7–10
7	5	7	2–3, 2–5, 2–6, 2–8, 3–6, 5–8, 6–8
8	3	2	1–3, 1–7
10	3	2	2–3, 2–5

The two nodes that create the fewest fills are nodes 4 and 8. Both nodes have the same number of degrees; therefore, the node with the lowest natural ordering, node 4, is chosen and eliminated.

The Scheme II algorithm continues until all nodes have been added to the ordering scheme and subsequently eliminated. Scheme II results in the following ordering:

$$\text{Ordering II} = [9\ 6\ 4\ 8\ 2\ 1\ 3\ 5\ 7\ 10]$$

Reordering the matrix of Example 4.6 to reflect the ordering of Scheme II yields the ordering with fills shown in Figure 4.19. This ordering yields only 10 fills, leading to an $\alpha = 84, \beta = 54$, and $\alpha + \beta = 138$. This represents a computational e ort of only 68% of the original unordered system.

Scheme I endeavors to reduce the number of multiplications and divisions in the LU factorization process, whereas Scheme II focuses on reducing the multiplications and divisions in the forward/backward substitution process. Scheme 0 o ers simplicity and speed of generation, but the performance improvement of Scheme I o sets the additional algorithm complexity [50]. Scheme II, however, frequently does not o er enough of an improvement to

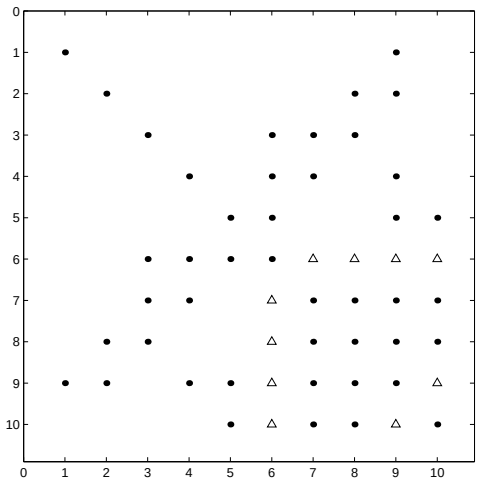


FIGURE 4.19
Matrix with fills for Example 4.9

merit implementation. The decision of which scheme to implement is problem dependent and is best left up to the user.

4.3.4 Other Schemes

Modifications to these algorithms have been introduced to reduce computational requirements. These modifications are summarized below [18]. The first modification to the minimum-degree algorithm is the use of mass elimination, inspired by the concept of indistinguishable nodes [17]. This modification allows a subset of nodes to be eliminated at one time. If two nodes x and y satisfy

$$Adj(y) \quad \{y\} = Adj(x) \quad \{x\} \tag{4.3}$$

where *Adj(y)* indicates the set of nodes adjacent to y, then nodes x and y are said to be indistinguishable and can be numbered consecutively in the ordering. This also reduces the number of nodes to be considered in an ordering, since only a representative node from each set of indistinguishable nodes needs to be considered. This accelerates the degree update step of the minimum-degree algorithm, which is typically the most computationally intensive step. Using mass elimination, the degree update is required only for the representative nodes.

The idea of incomplete degree update allows avoiding degree update for nodes that are known not to be minimum degree. Between two nodes u and v, node v is said to be outmatched by u if [11]

$$Adj(u) \quad \{u\} \quad Adj(v) \quad \{v\} \tag{4.4}$$

Thus, if a node v becomes outmatched by u in the elimination process, the node u can be eliminated before v in the minimum-degree ordering algorithm. From this, it follows that it is not necessary to update the degree of v until node u has been eliminated. This further reduces the time-consuming degree update steps.

Another modification to the minimum-degree algorithm is one in which all possible nodes of minimum degree are eliminated before the degree update step. At a given step in the elimination process, the elimination of node y does not change the structure of the remaining nodes not in *Adj(y)*. The multiple-minimum-degree (MMD) algorithm delays degree update of the nodes in *Adj(y)*, and chooses another node with the same degree as y to eliminate. This process continues until there are no more nodes left with the same degree as y. This algorithm was found to perform as well as the minimum-degree algorithm regarding the number of fills introduced [30]. In addition, it was found that the MMD algorithm performed faster. This was attributed to the identification of indistinguishable and outmatched nodes earlier in the algorithm, as well as the reduced number of degree updates.

Ties often occur for a given criteria (degrees or fills) in an ordering algorithm. The tie breaker often falls back on the natural ordering of the original matrix. It has been recognized that the natural ordering greatly a ects the factorization in terms of number of fills and computation time. Thus it is often preferable to use a rapid "pre-conditioning" ordering before applying the ordering algorithm. Scheme 0 o ers one such pre-ordering, but to date no consistent optimum method for pre-ordering has been developed that works well for all types of problems.

4.4 Power System Applications

Large sparse matrices occur frequently in power system applications, including state estimation, power flow analysis, and transient and dynamic stability simulations. Computational e ciency of these applications depends heavily on their formulation and the use of sparse matrix techniques. To better understand the impact of sparsity on power system problems, consider the power flow Jacobian of the IEEE 118 bus system shown in Figure 4.20.

The Jacobian of this system has 1051 non-zero elements and has the structure shown in Figure 4.21(a). Note the dominance of the main diagonal and then the two sub-diagonals which result from the $\frac{\partial \Delta Q}{\partial \delta}$ and $\frac{\partial \Delta P}{\partial V}$ sub-Jacobians. The LU factorization of this Jacobian yields the structure shown in Figure 4.21(b). This matrix has 14849 non-zero elements. Notice that the two sub-diagonals have created a large number of fills extending between them and the main diagonal.

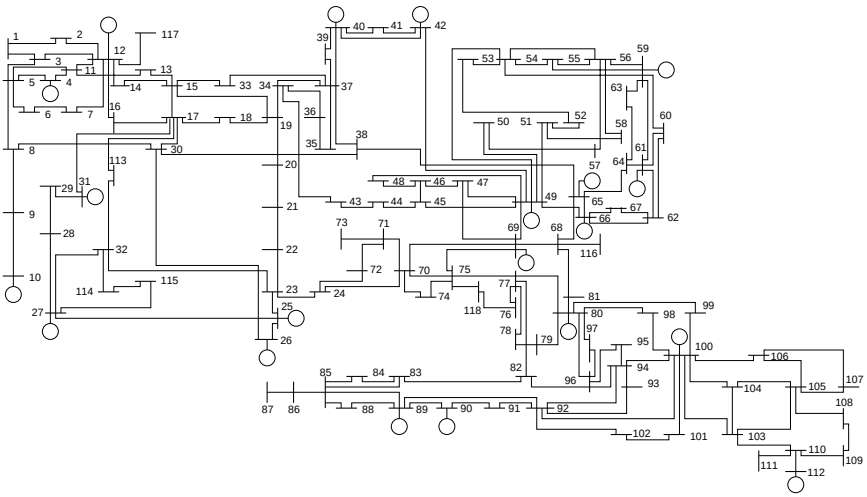
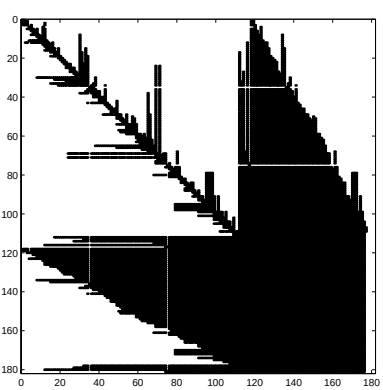


FIGURE 4.20
IEEE 118 bus system



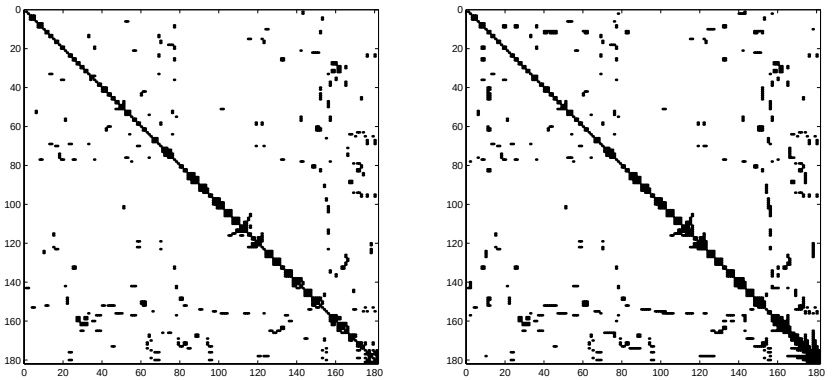


FIGURE 4.24
IEEE 118 bus system Scheme 2 (a) Jacobian, and (b) LU factors

Figure 4.22(a) shows the structure of the load flow Jacobian reordered according to Scheme 0. In this reordering, the presence of the sub-diagonals is gone. The LU factorization of the Scheme 0 reordered Jacobian yields the structure shown in Figure 4.22(b). This matrix has only 1869 non-zero elements, which is almost an order of magnitude reduction from the non-ordered Jacobian.

Figure 4.23(a) shows the structure of the load flow Jacobian reordered according to Scheme 1. Note how the elements are gradually pulling into the main diagonal, which leads to a decrease in the number of fills. The LU factorization of the Scheme 1 ordering is shown in Figure 4.23(b), which has 1455 non-zero elements.

Lastly, Figure 4.24(a) shows the structure of the Scheme 2 reordered Jacobian which yields the LU factorization in Figure 4.24(b). This ordering yields only 1421 non-zero elements, which is more than a full order of magnitude reduction. The LU factorization solution time for a sparse matrix is on the order of n^2 multiplications and divisions. The non-reordered load flow solution would require on the order of 220.5×10^6 multiplications and divisions per iteration, whereas the Scheme 2 reordered load flow solution would require only 2.02×10^6 multiplications and divisions. Thus, the solution of the reordered system is over 100 times faster than the original system! When the solution time is multiplied by the number of iterations in a Newton-Raphson power flow or by the number of time steps in a time-domain integration, it would be computationally foolhardy to not use a reordering scheme.

4.5 Problems

1. Verify equations (4.1) and (4.2) for calculating α and β.
2. Let A and B be two sparse (square) matrices of the same dimension. How can the graph of $C = A + B$ be characterized with respect to the graphs of A and B?
3. Consider the matrix

$$A = \begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}$$

   (a) Draw the graphical representation of A. How many multiplications and divisions will the LU factorization of A require?

   (b) Reorder the matrix using the new node numbering $\phi = [1, 3, 4, 2, 5, 6]$. Draw the graphical representation of the reordered matrix. How many multiplications and divisions will the LU factorization of the reordered matrix require?

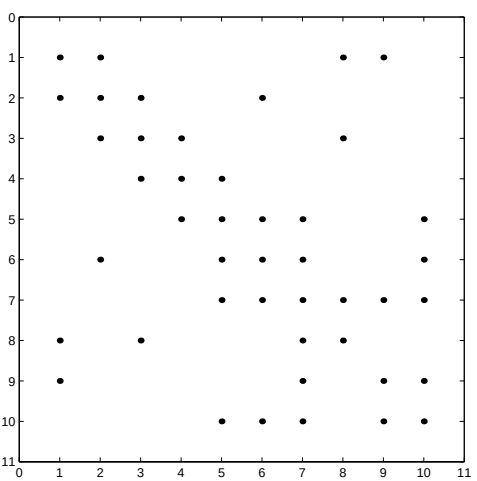


FIGURE 4.25
Sparse Test System I

In general, the Taylor series-based integration methods can be expressed as

$$x_{n+1} = x_n + hT_p(x_n) \tag{5.3}$$

where

$$T_p(x_n) = f(x(t_n), t_n) + \frac{h^2}{2!} f'(x(t_n), t_n) + \ldots + \frac{h^p}{p!} f^{(p-1)}(x(t_n), t_n)$$

and the integer p is called the *order* of the integration method. This method is very accurate for large p, but is not computationally e cient for large p since it requires a large number of function derivatives and evaluations.

5.1.2 Forward-Euler Method

For $p = 1$, the Taylor series-based integration algorithm is given by:

$$x_{n+1} = x_n + hf(x_n, t_n) \tag{5.4}$$

which is also the well-known *Euler* or *forward Euler* method.

5.1.3 Runge-Kutta Methods

A second order Taylor's method can be derived for $p = 2$.

$$\begin{aligned} x_{n+1} &= x_n + hT_2(x_n, t_n) \\ &= x_n + hf(x_n, t_n) + \frac{h^2}{2} f'(x_n, t_n) \end{aligned}$$

As the order of the Taylor's method increases, so does the number of derivatives and partial derivatives. In many cases, the analytic derivation of the derivatives can be replaced by a numerical approximation. One of the most commonly known higher-order Taylor series-based integration methods is the Runge-Kutta method, where the derivatives are replaced by approximations. The fourth-order Runge-Kutta method is given by

$$x_{n+1} = x_n + hK_4(x_n, t_n) \tag{5.5}$$

where K_4 is an approximation to T_4:

$$\begin{aligned} K_4 &= \frac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4] \\ k_1 &= f(x_n, t_n) \\ k_2 &= f\left(x_n + \frac{h}{2}k_1, t_n + \frac{h}{2}\right) \\ k_3 &= f\left(x_n + \frac{h}{2}k_2, t_n + \frac{h}{2}\right) \\ k_4 &= f(x_n + hk_3, t_n + h) \end{aligned}$$

where each k_i represents the slope (derivative) of the function at four di erent points. The slopes are then weighted $\begin{bmatrix} 1 & 2 & 2 & 1 \\ 6 & 6 & 6 & 6 \end{bmatrix}$ to approximate the T_4 function.

The advantages of Taylor series-based methods is that the method is straightforward to program and only depends on the previous time step. These methods (especially the Runge-Kutta methods) su er from di cult error analysis, however, since the derivatives are approximated and not found analytically. Therefore the integration step size is typically chosen conservatively (small), and computational e ciency may be lost.

5.2 Multistep Methods

Another approach to approximating the solution $x(t)$ of equation (5.1) is to approximate the nonlinear function as a polynomial of degree k such that

$$\hat{x}(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \ldots + \alpha_k t^k \tag{5.6}$$

where the coe cients $\alpha_0, \alpha_1, \ldots, \alpha_k$ are constant. It can be proven that any function can be approximated arbitrarily closely (within a pre-determined ε) with a polynomial of su ciently high degree on a finite interval $[t_0, t_N]$. The polynomial approximation can be related to the solution of equation (5.1) through the introduction of *multistep* methods. A multistep method is one in which the approximation x_{n+1} can be a function of any number of previous numerical approximations $x_n, x_{n-1}, \ldots$ and corresponding functions $f(x_n, t_n), f(x_{n-1}, t_{n-1}), \ldots$ unlike one-step methods (such as the Runge-Kutta) which depend only on the information from the immediately previous step. In general,

$$\begin{aligned} x_{n+1} &= a_0 x_n + a_1 x_{n-1} + \ldots + a_p x_{n-p} + h[b_{-1} f(x_{n+1}, t_{n+1}) + b_0 f(x_n, t_n) \\ &\quad + b_1 f(x_{n-1}, t_{n-1}) + \ldots + b_p f(x_{n-p}, t_{n-p})] \qquad (5.7) \\ &= \sum_{i=0}^{p} a_i x_{n-i} + h \sum_{i=-1}^{p} b_i f(x_{n-i}, t_{n-i}) \qquad (5.8) \end{aligned}$$

To relate the integration method to the polynomial approximation, a relationship between the coe cients must be determined. A k-degree polynomial is uniquely determined by $k + 1$ coe cients $(\alpha_0, \ldots, \alpha_k)$. The numerical integration method has $2p + 3$ coe cients; therefore, the coe cients must be chosen such that

$$2p + 3 \geq k + 1 \tag{5.9}$$

The order of the numerical integration method is the highest degree k of a polynomial in t for which the numerical solution coincides with the exact

solution. The coe cients may be determined by selecting a set of linear basis functions $[\phi_1(t)\ \phi_2(t),\ \ldots,\ \phi_k(t)]$ such that

$$\phi_j(t) = t^j \quad j = 0, 1, \ldots, k$$

and solving the set of multistep equations

$$\phi_j\,(t_{n+1}) = \sum_{i=0}^{p} a_i \phi_j\,(t_{n-i}) + h_{n+1}\left[\sum_{i=-1}^{p} b_i \dot{\phi}\,(t_{n-i})\right]$$

for all $j = 0, 1, \ldots, k$.

This method can be applied to derive several first order numerical integration methods. Consider the case where $p = 0$ and $k = 1$. This satisfies the constraint of equation (5.9); thus, it is possible to determine multistep coefficients that will result in an exact polynomial of degree 1. The set of basis functions for $k = 1$ is

$$\phi_0(t) = 1 \tag{5.10}$$

$$\phi_1(t) = t \tag{5.11}$$

which lead to the derivatives

$$\dot{\phi}_0(t) = 0 \tag{5.12}$$

$$\dot{\phi}_1(t) = 1 \tag{5.13}$$

and the multistep equation

$$x_{n+1} = a_0 x_n + b_{-1} h_{n+1} f\,(x_{n+1}, t_{n+1}) + b_0 h_{n+1} f\,(x_n, t_n) \tag{5.14}$$

Representing the multistep method of equation (5.14) in terms of basis functions yields the following two equations

$$\phi_0\,(t_{n+1}) = a_0 \phi_0(t_n) + b_{-1} h_{n+1} \dot{\phi}_0\,(t_{n+1}) + b_0 h_{n+1} \dot{\phi}_0\,(t_n) \tag{5.15}$$

$$\phi_1\,(t_{n+1}) = a_0 \phi_1(t_n) + b_{-1} h_{n+1} \dot{\phi}_1\,(t_{n+1}) + b_0 h_{n+1} \dot{\phi}_1\,(t_n) \tag{5.16}$$

Substituting the choice of basis functions of equations (5.10) and (5.11) into equations (5.15) and (5.16) results in

$$1 = a_0(1) + b_{-1} h_{n+1}(0) + h_{n+1} b_0(0) \tag{5.17}$$

$$t_{n+1} = a_0 t_n + b_{-1} h_{n+1}(1) + b_0 h_{n+1}(1) \tag{5.18}$$

From equation (5.17), the coe cient $a_0 = 1$. Recalling that $t_{n+1} - t_n = h_{n+1}$, equation (5.18) yields

$$b_{-1} + b_0 = 1 \tag{5.19}$$

This choice of order and degree leads to two equations in three unknowns; therefore, one of them may be chosen arbitrarily. By choosing $a_0 = 1, b_{-1} = 0$, and

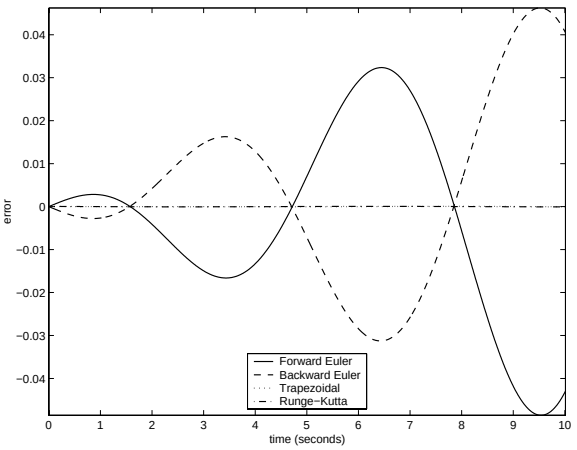


FIGURE 5.2
Error in numerical solutions for Example 5.1

whose magnitude is slightly less than the exact solution and is decreasing with time. Both properties are due to the *local truncation error* of the methods. The forward-Euler method has a tendency to generate numerical solutions that increase with time (under-damped), whereas the backward-Euler method tends to add damping to the numerical solution. Therefore, caution must be used when using either of these first-order methods for numerical integration.

Figure 5.2 shows the global error with time for each of the numerical methods. Note that the errors for the forward and backward-Euler methods are equal in magnitude, but opposite in sign. This relationship will be further discussed later in this chapter. The numerical errors for the trapezoidal and the Runge-Kutta methods are reproduced in Figure 5.3 on a magnified scale. The errors in each method are comparable even though the trapezoidal method is a second order polynomial approximation method and the Runge-Kutta is a fourth order Taylor method. Section 5.3 will further explore the development of expressions for estimating the error of various integration methods.

When implicit methods, such as the trapezoidal method, are used to solve nonlinear systems of di erential equations, the system of equations must be solved iteratively at each time step. For example, consider the following nonlinear system of equations:

$$\dot{x} = f(x(t), t) \quad x_0 = x(t_0) \tag{5.47}$$

Applying the trapezoidal method to numerically integrate this system results in the following discretized system:

$$x_{n+1} = x_n + \frac{h}{2}\left[f\left(x_n, t_n\right) + f\left(x_{n+1}, t_{n+1}\right)\right] \tag{5.48}$$

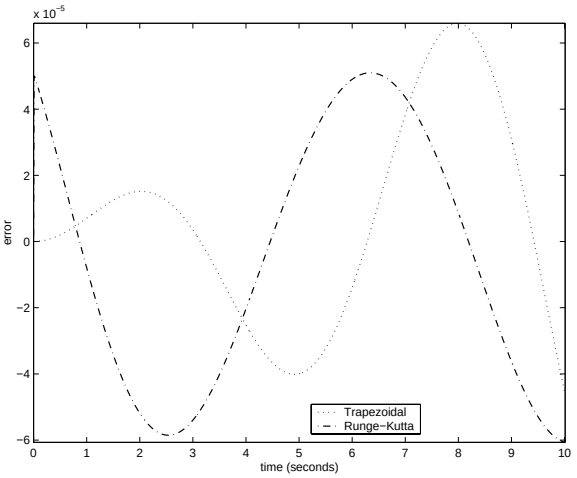


FIGURE 5.3
Error in trapezoidal and Runge-Kutta numerical solutions for Example 5.1

Since this nonlinear expression is implicit in x_{n+1}, it must be solved numerically:

$$x_{n+1}^{k+1} = x_{n+1}^{k} - \left[I - \frac{h}{2}\frac{\partial f}{\partial x}\right]^{-1}\Bigg|_{x_{n+1}^{k}} \left(x_{n+1}^{k} - x_n - \frac{h}{2}\left[f(x_n) + f\left(x_{n+1}^{k}\right)\right]\right) \tag{5.49}$$

where k is the Newton-Raphson iteration index, I is the identity matrix, and x_n is the converged value from the previous time step.

5.2.1 Adams Methods

Recall that the general class of multistep methods may be represented by

$$x_{n+1} = \sum_{i=0}^{p} a_i x_{n-i} + h \sum_{i=-1}^{p} b_i f(x_{n-i}, t_{n-i}) \tag{5.50}$$

A numerical multistep algorithm will give the exact value for x_{n+1} if $x(t)$ is a polynomial of degree less than or equal to k if the following *exactness constraints* are satisfied:

$$\sum_{i=0}^{p} a_i = 1 \tag{5.51}$$

$$\sum_{i=1}^{p} (-i)^p a_i + j \sum_{i=-1}^{p} (-i)^{(j-1)} b_i = 1 \quad \text{for } j = 1, 2, \ldots, k \tag{5.52}$$

The exactness constraint of equation (5.51) is frequently referred to as the *consistency* constraint. Numerical multistep integration algorithms that satisfy equation (5.51) are said to be "consistent." For a desired polynomial of degree k, these constraints can be satisfied by a wide variety of possibilities. Several families of methods have been developed by pre-defining some of the relationships between the coe cients. The family of *Adams* methods are defined by setting the coe cients $a_1 = a_2 = \ldots = a_p = 0$. By the consistency constraint, the coe cient a_0 must therefore equal 1.0. Thus, the Adams methods are reduced to

$$x_{n+1} = x_n + h \sum_{i=-1}^{p} b_i f(x_{n-i}, t_{n-i}) \tag{5.53}$$

where $p = k - 1$. The Adams methods can be further classified by the choice of implicit or explicit integration. The explicit class, frequently referred to as the "Adams-Bashforth" methods, is specified by setting $b_{-1} = 0$ and applying the second exactness constraint as:

$$\sum_{i=0}^{k-1} (-i)^{(j-1)} b_i = \frac{1}{j} \quad j = 1, \ldots, k \tag{5.54}$$

In matrix form, equation (5.54) becomes

$$\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 0 & -1 & -2 & \ldots & -(k-1) \\ 0 & 1 & 4 & \ldots & (-(k-1))^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & (-1)^{(k-1)} & (-2)^{(k-1)} & \ldots & (-(k-1))^{(k-1)} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{k-1} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \vdots \\ \frac{1}{k} \end{bmatrix} \tag{5.55}$$

By choosing the desired degree k (and subsequently the order p), the remaining b_i coe cients may be found from solving equation (5.55).

Example 5.2

Find the third-order Adams-Bashford integration method.

Solution 5.2 Setting $k = 3$ yields the following linear system:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -2 \\ 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \end{bmatrix}$$

Solving this system yields

$$b_0 = \frac{23}{12}$$

$$b_1 = -\frac{16}{12}$$

$$b_2 = \frac{5}{12}$$

Thus the third-order Adams-Bashforth method is given by:

$$x_{n+1} = x_n + \frac{1}{12} h\left[23f(x_n, t_n) - 16f(x_{n-1}, t_{n-1}) + 5f(x_{n-2}, t_{n-2})\right] \quad (5.56)$$

When implementing this algorithm, the values of x_n, x_{n-1}, and x_{n-2} must be saved in memory.

The implicit versions of the Adams methods have $b_{-1} = 0$, $p = (k-2)$, are called the "Adams-Moulton" methods, and are given by

$$x_{n+1} = x_n + h \sum_{i=-1}^{k-2} b_i f(x_{n-i}, t_{n-i}) \quad (5.57)$$

The second exactness constraint yields

$$\sum_{i=-1}^{k-2} (-i)^{(j-1)} b_i = \frac{1}{j} \quad j = 1, \ldots, k \quad (5.58)$$

or in matrix form:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & 0 & -1 & -2 & \ldots & -(k-2) \\ 1 & 0 & 1 & 4 & \ldots & (-(k-2))^2 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & (-1)^{(k-2)} & (-2)^{(k-2)} & \ldots & (-(k-2))^{(k-2)} \end{bmatrix} \begin{bmatrix} b_{-1} \\ b_0 \\ b_1 \\ \vdots \\ b_{k-2} \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \vdots \\ \frac{1}{k} \end{bmatrix} \quad (5.59)$$

Example 5.3
Find the third-order Adams-Moulton integration method.

Solution 5.3 Setting $k = 3$ yields the following linear system:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_{-1} \\ b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \end{bmatrix}$$

Solving this system yields

$$b_{-1} = \frac{5}{12}$$
$$b_0 = \frac{8}{12}$$
$$b_1 = -\frac{1}{12}$$

Thus the third-order Adams-Moulton method is given by:

$$x_{n+1} = x_n + \frac{1}{12} h\left[5f(x_{n+1}, t_{n+1}) + 8f(x_n, t_n) - f(x_{n-1}, t_{n-1})\right] \quad (5.60)$$

When implementing this algorithm, the values of x_n and x_{n-1} must be saved in memory and the equations must be solved iteratively if the function $f(x)$ is nonlinear.

The Adams-Moulton method is implicit and must be solved iteratively using the Newton-Raphson method (or other similar method) as shown in equation (5.49). Iterative methods require an initial value for the iterative process to reduce the number of required iterations. The explicit Adams-Bashforth method is frequently used to estimate the initial value for the implicit Adams-Moulton method. If su ciently high-order predictor methods are used, the Adams-Moulton method iteration will typically converge in only one iteration. This process is often called a *predictor-corrector* approach; the Adams-Bashforth method predicts the solution and the implicit Adams-Moulton corrects the solution.

Another implementation issue for multistep methods is how to start up the integration at the beginning of the simulation since a high order method requires several previous values. The usual procedure is to use a high-order one-step method or to increase the number of steps of the method with each time step to generate the required number of values for the desired multistep method.

5.2.2 Gear's Methods

Another well-known family of multistep methods are the Gear's methods [14]. This family of methods is particularly well suited for the numerical solution of sti systems. As opposed to the Adams family of methods where all the a_i coe cients except a_0 are zero, Gear's methods are identified by having all of the b_i coe cients equal to zero except b_{-1}. Obviously since $b_{-1} = 0$, all Gear's methods are implicit methods. The k-th order Gear's algorithm is obtained by setting $p = k - 1$ and $b_0 = b_1 = \ldots = 0$ yielding

$$x_{n+1} = a_0 x_n + a_1 x_{n-1} + \ldots + a_{k-1} x_{n-k+1} + h b_{-1} f(x_{n+1}, t_{n+1}) \qquad (5.61)$$

The $k + 1$ coe cients can be calculated explicitly by applying the exactness constraints as illustrated with the Adams methods:

$$\begin{bmatrix} 1 & 1 & 1 \ldots & 1 & 0 \\ 0 & -1 & -2 \ldots & -(k-1) & 1 \\ 0 & 1 & 4 \ldots & [-(k-1)]^2 & 2 \\ \vdots & \vdots & \vdots \ddots & \vdots & \vdots \\ 0 & (-1)^k & (-2)^k \ldots & [-(k-1)]^k & k \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ b_{-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \qquad (5.62)$$

The solution of equation (5.62) uniquely determines the $k + 1$ coe cients of the k-th order Gear's method.

Example 5.4
Find the third-order Gear's integration method.

Solution 5.4 Setting $k = 3$ yields the following linear system:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 1 & 4 & 2 \\ 0 & -1 & -8 & 3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ b_{-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Solving this system yields

$$b_{-1} = \frac{6}{11}$$
$$a_0 = \frac{18}{11}$$
$$a_1 = -\frac{9}{11}$$
$$a_2 = \frac{2}{11}$$

Thus the third-order Gear's method is given by:

$$x_{n+1} = \frac{18}{11}x_n - \frac{9}{11}x_{n-1} + \frac{2}{11}x_{n-2} + \frac{6}{11}hf(x_{n+1}, t_{n+1}) \tag{5.63}$$

When implementing this algorithm, the values of x_n through x_{n-2} must be saved in memory and the equations must be solved iteratively if the function $f(x)$ is nonlinear.

5.3 Accuracy and Error Analysis

The accuracy of numerical integration methods is impacted by two primary causes: computer round-o error and truncation error. Computer round-o error occurs as a result of the finite precision of the computer upon which the algorithm is implemented and little can be done to reduce this error short of using a computer with greater precision. A double precision word length is normally used for scientific computation. The di erence between the exact solution and the calculated solution is dominated by truncation error, which arises from the truncation of the Taylor series or polynomial being used to approximate the solution.

In the implementation of numerical integration algorithms, the most e ective methods are those methods that require the least amount of calculation to yield the most accurate results. In general, higher order methods produce the

most accurate results, but also require the greatest amount of computation. Therefore, it is desirable to compute the solution as infrequently as possible by taking the largest time step possible between intervals. Several factors impact the size of the time step including the error introduced at each step by the numerical integration process itself. This error is the *local truncation error* (LTE) and arises from the truncation of the polynomial approximation and/or the truncation of the Taylor series expansion depending on the method used. The term *local* emphasizes that the error is introduced locally and is not residual global error from earlier time steps. The error introduced at a single step of an integration method is given by

$$\varepsilon_T \triangleq x(t_{n+1}) - x_{n+1} \tag{5.64}$$

where $x(t_{n+1})$ is the exact solution at time t_{n+1} and x_{n+1} is the numerical approximation. This definition assumes that this is the error *introduced in one step*, therefore $x(t_n) = x_n$. The local truncation error is shown graphically in Figure 5.4. To compute the error, the solution $x(t_{n-i})$ is expanded about t_{n+1}:

$$x_{n-i} = x(t_{n-i}) = \sum_{j=0}^{\infty} \frac{(t_{n-i} - t_{n+1})^j \, d^{(j)}}{j!}$$

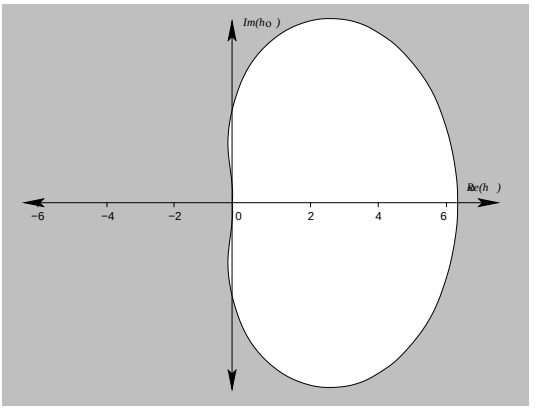


FIGURE 5.7
Region of absolute stability for Gear's third-order method

Adams-Moulton

The region of absolute stability of the Adams-Moulton methods can be developed from equation (5.100) by setting $p = k - 1$, and $a_1, a_2, \ldots = 0$:

$$h\lambda(\theta) = \frac{e^{jk\theta} - a_0 e^{j(k-1)\theta}}{b_{-1}e^{jk\theta} + b_0 e^{j(k-1)\theta} + b_1 e^{j(k-2)\theta} + \ldots + b_{k-2}e^{j\theta}} \tag{5.103}$$

After substituting in the third-order coefficients, the expression for the region of absolute stability as a function of θ is given by

$$h\lambda(\theta) = \frac{e^{j3\theta} - e^{j2\theta}}{\frac{5}{12}e^{j3\theta} + \frac{8}{12}e^{j2\theta} - \frac{1}{12}e^{j\theta}} \tag{5.104}$$

The region of absolute stability of the Adams-Moulton third-order method is shown as the shaded region of Figure 5.8.

Adams-Bashforth

The stability of the family of Adams-Bashforth methods can be derived from equation (5.100) by setting $p = k - 1$, $b_{-1} = 0$, and $a_1, a_2, \ldots, = 0$:

$$h\lambda(\theta) = \frac{e^{jk\theta} - a_0 e^{j(k-1)\theta}}{b_0 e^{j(k-1)\theta} + b_1 e^{j(k-2)\theta} + \ldots + b_{k-1}} \tag{5.105}$$

$$h\lambda(\theta) = \frac{e^{j3\theta} - e^{j2\theta}}{\frac{23}{12}e^{j2\theta} - \frac{16}{12}e^{j\theta} + \frac{5}{12}} \tag{5.106}$$

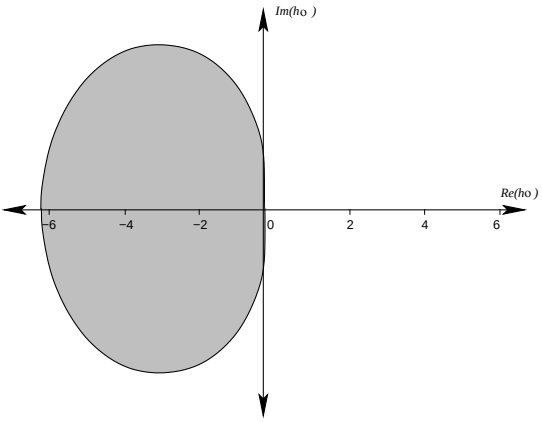


FIGURE 5.8
Region of absolute stability for Adams-Moulton third-order method

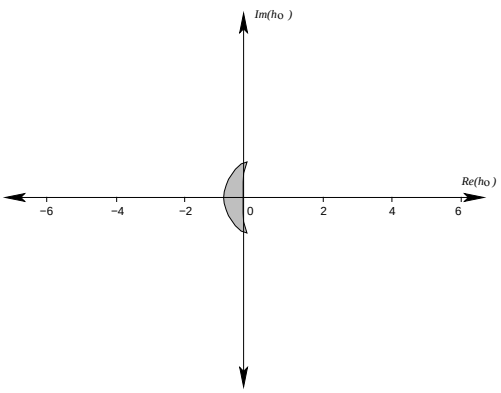


FIGURE 5.9
Region of absolute stability for Adams-Bashforth third-order method

The region of absolute stability of the Adams-Bashforth method is shown as the shaded region in Figure 5.9.

This example illustrates one of the primary di erences between implicit and explicit methods. For the same order, the two implicit methods (Gear's and Adams-Moulton) exhibit much larger regions of absolute stability than does the explicit method (Adams-Bashforth). The Gear's region of absolute stability contains nearly the entire left half $h\lambda$ plane; thus, for any stable dynamic system, the step size can be chosen as large as desired without any consideration for numerical stability. Even the Adams-Moulton region of absolute stability is quite large compared to the Adams-Bashforth. Typically, the region of stability of explicit methods is much smaller than the region of absolute stability of corresponding order implicit methods. For this reason, implicit methods are frequently used in commercial integration packages so that the integration step size can be chosen based purely on the local truncation error criteria. The region of absolute stability shrinks as the order of the method increases, whereas the accuracy increases. This is a trade-o between accuracy, stability, and numerical e ciency in integration algorithms.

5.5 Stiff Systems

Gear's methods were originally developed for the solution of systems of *sti* ordinary di erential equations. Sti systems are systems that exhibit a wide-range of time varying dynamics from "very fast" to "very slow." A sti linear system exhibits eigenvalues that span several orders of magnitude. A nonlinear system is sti if its associated Jacobian matrix exhibits widely separated eigenvalues when evaluated at the operating points of interest. For e cient and accurate solutions of sti di erential equations, it is desirable for a multistep method to be "sti y stable." An appropriate integration algorithm will allow the step size to be varied over a wide range of values and yet will remain numerically stable. A sti y stable method exhibits the three stability regions shown in Figure 5.10 such that:

1. Region I is a region of absolute stability
2. Region II is a region of accuracy and stability
3. Region III is a region of accuracy and relative stability

Only during the initial period of the solution of the ODE do the large negative eigenvalues significantly impact the solution, yet they must be accounted for throughout the whole solution. Large negative eigenvalues ($\lambda < 0$) will decay rapidly by a factor of $1/e$ in time $1/\lambda$. If $h\lambda = \gamma + j\beta$, then the change in magnitude in one step is e^{γ}. If $\gamma \leq \delta \leq 0$, where δ defines the interface

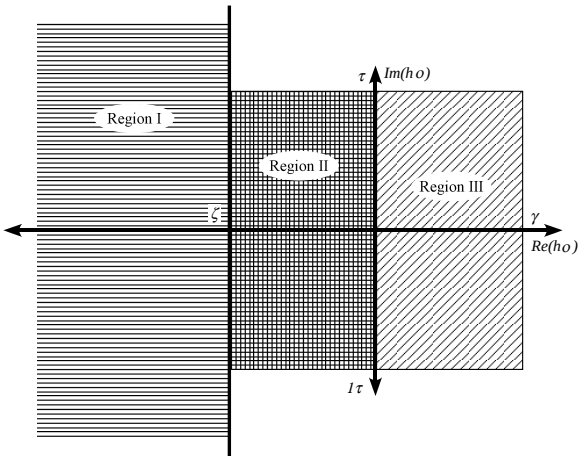


FIGURE 5.10
Regions required for sti stability

between Regions I and II, then the component is reduced by at least e^{δ} in one step. After a finite number of steps, the impact of the fast components is negligible and their numerical accuracy is unimportant. Therefore the integration method is required to be absolutely stable in Region I.

Around the origin, numerical accuracy becomes more significant and relative or absolute stability is required. A region of *relative stability* consists of those values of $h\lambda$ for which the extraneous eigenvalues of the characteristic polynomial of equation (5.97) are less in magnitude than the principal eigenvalue. The principal eigenvalue is the eigenvalue which governs the system response most closely. If the method is relatively stable in Region III, then the system response will be dominated by the principal eigenvalue in that Region. If $\gamma > \alpha > 0$, one component of the system response is increasing by at least e^{α} one step. This increase must be limited by choosing the step sizes small enough to track this change.

If $\beta > \theta$, there are at least $\theta/2\pi$ complete cycles of oscillation in one step. Except in Region I where the response is rapidly decaying, and where $\gamma > \alpha$ is not used, the oscillatory responses must be captured. In practice, it is customary to have eight or more time points per cycle (to accurately capture the magnitude and frequency of the oscillation); thus, θ is chosen to be bounded by $\pi/4$ in Region II.

Examination of the family of Adams-Bashforth methods shows that they all fail to satisfy the criteria to be sti y stable and are not suitable for integrating sti systems. Only the first and second order Adams-Moulton (backward-Euler and trapezoidal, respectively) satisfy the sti y stable criteria. Gear's algorithms on the other hand were developed specifically to

address sti system integration [14]. Gear's algorithms up to order six satisfy the sti properties with the following choice of δ [6]:

Order	δ
1	0
2	0
3	0.1
4	0.7
5	2.4
6	6.1

Example 5.7

Compare the application of the third-order Adams-Bashforth, Adams-Moulton, and Gear's method to the integration of the following system:

$$\dot{x}_1 = 48x_1 + 98x_2 \quad x_1(0) = 1 \tag{5.107}$$

$$\dot{x}_2 = -49x_1 - 99x_2 \quad x_2(0) = 0 \tag{5.108}$$

Solution 5.7 The exact solution to Example 5.7 is

$$x_1(t) = 2e^{-t} - e^{-50t} \tag{5.109}$$

$$x_2(t) = -e^{-t} + e^{-50t} \tag{5.110}$$

This solution is shown in Figure 5.11. Both states contain both fast and slow components, with the fast component dominating the initial response and the slow component dominating the longer term dynamics. Since the Gear's, Adams-Bashforth, and Adams-Moulton methods are multistep methods, each method is initialized using the absolutely stable trapezoidal method for the first two to three steps using a small step size.

The Adams-Bashforth algorithm with a time step of 0.0111 seconds is shown in Figure 5.12. Note that even with a small step size of 0.0111 seconds the inherent error in the integration algorithm eventually causes the system response to exhibit numerical instabilities. The step size can be decreased to increase the stability properties, but this requires more time steps in the integration window (t [0, 2]) than is computationally necessary.

The Adams-Moulton response to the sti system for an integration step size of 0.15 seconds is shown in Figure 5.13. Although a much larger time step can be used for integration as compared to the Adams-Bashforth algorithm, the Adams-Moulton algorithm does not exhibit numerical absolute stability. For an integration step size of $h = 0.15$ seconds, the Adams-Moulton algorithm exhibits numerical instability. Note that the solution is oscillating with growing magnitude around the exact solution.

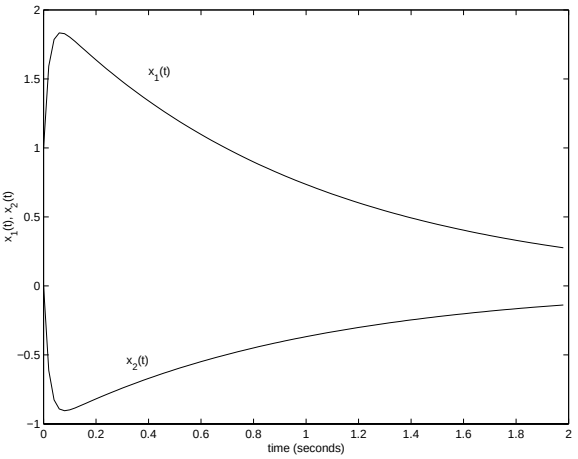


FIGURE 5.11
Sti system response

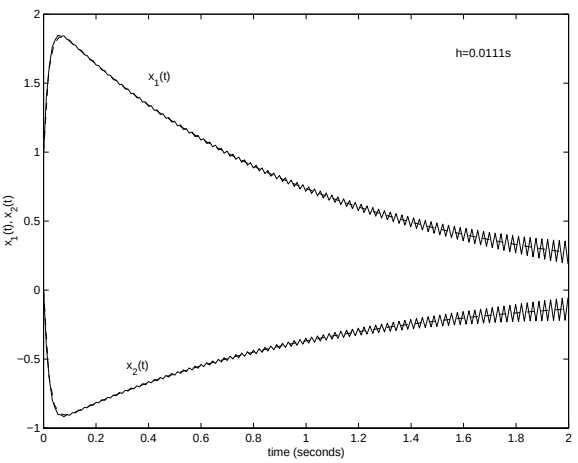


FIGURE 5.12
Adams-Bashforth sti system response with h = 0.0111 s step size

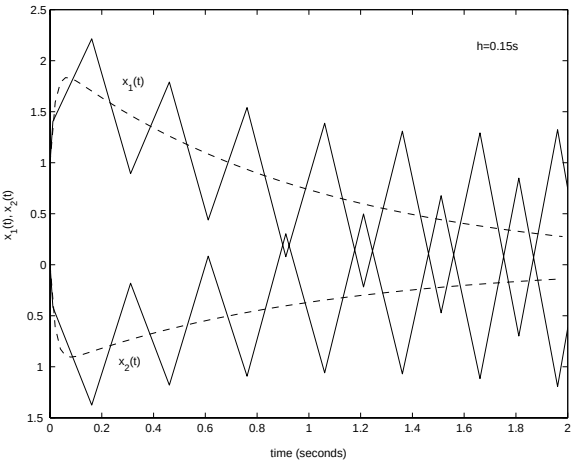


FIGURE 5.13

5.6 Step-Size Selection

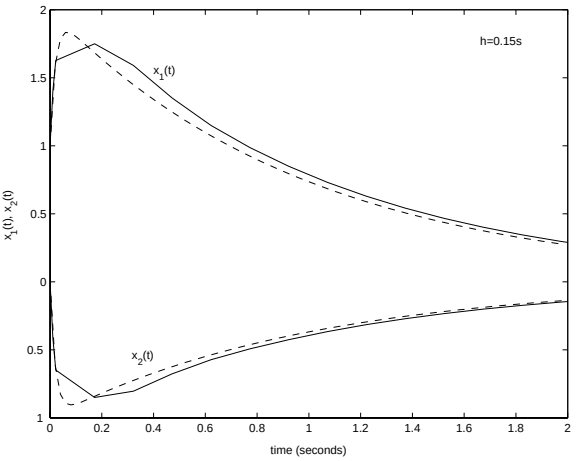


FIGURE 5.14
Gear•s sti system response with $h = 0.15$ s step size

and latency. In this case, it is desirable for the integration step size to have the ability to increase or decrease throughout the simulation interval. This can be accomplished by choosing the integration step size based on the local truncation error bounds.

Consider the trapezoidal method absolute local truncation error:

$$\tau = \frac{1}{12} h^3 x^{(3)}(\ ) \tag{5.111}$$

The local truncation error is dependent on the integration step size h and the third derivative of the function $x^{(3)}(\ )$. If the local truncation error is chosen to be in the interval:

$$B_L \quad B_U \tag{5.112}$$

where B_L and B_U represent the prespeci“ed lower and upper bounds respectively, then the integration step size can be bounded by

$$h \quad \sqrt[3]{\frac{12 B_U}{x^{(3)}(\ )}} \tag{5.113}$$

If $x(t)$ is rapidly varying, the $x^{(3)}(\ )$ will be large and h must be chosen small to satisfy B_U, whereas if $x(t)$ is not varying rapidly, then $x^{(3)}(\ )$ will be small and h can be chosen relatively large while still satisfying B_U. This leads to the following procedure for calculating the integration step size:

Integration Step Size Selection

Attempt an integration step size h_{n+1} to calculate x_{n+1} from $x_n, x_{n-1}, \ldots$

1. Using x_{n+1}, calculate the local truncation error τ.
2. If $B_L \quad \tau \quad B_U$, then PASS, accept h_{n+1}, $h_{next} = h_{n+1}$, and continue.
3. If $> B_U$, then FAIL (h_{n+1} is too large), set $h_{n+1} = h_{n+1}$, repeat integration for x_{n+1}.
4. If B_U, then PASS, accept h_{n+1}, set $h_{next} = h_{n+1}$, and continue.

where

$$= B_{avg}^{\frac{1}{k+1}} \tag{5.114}$$

where $B_L \quad B_{avg} \quad B_U$ and k is the degree of the method.

Some commercial integration packages implement an algorithm that is slightly different than this one. In these packages, if the local truncation error is smaller than the lower bound, then the attempted integration step size also FAILS, and the integration step is re-attempted with a larger integration step size. Once again, there is a trade off between the time spent in recalculating x_{n+1} with a larger step size and the additional computational effort acquired by simply accepting the current value and continuing on.

The difficulty in implementing this step size selection approach is the calculation of the higher order derivatives of $x(t)$. Since $x(t)$ is not known analytically, the derivatives must be calculated numerically. One common approach is to use difference methods to approximate the derivatives. The $(k+1)^{st}$ derivative to $x(\ )$ is approximated by

$$x^{(k+1)}(\ ) \quad (k+1)! \quad {}_{k+1}x_{n+1} \tag{5.115}$$

where ${}_{k+1}x_{n+1}$ is found recursively:

$${}_1x_{n+1} = \frac{x_{n+1} - x_n}{t_{n+1} - t_n} \tag{5.116}$$

$${}_1x_n = \frac{x_n - x_{n-1}}{t_n - t_{n-1}} \tag{5.117}$$

$$\vdots \tag{5.118}$$

$${}_2x_{n+1} = \frac{{}_1x_{n+1} - {}_1x_n}{t_{n+1} - t_{n-1}} \tag{5.119}$$

$${}_2x_n = \frac{{}_1x_n - {}_1x_{n-1}}{t_n - t_{n-2}} \tag{5.120}$$

$$\vdots \tag{5.121}$$

$${}_{k+1}x_{n+1} = \frac{{}_kx_{n+1} - {}_kx_n}{t_{n+1} - t_{n-k}} \tag{5.122}$$

Example 5.8
Find an expression for step size selection of the trapezoidal integration method with an upper bound of 10^{-3} on the local truncation error.

Solution 5.8
The LTE for the trapezoidal method is given by

$$h \quad \sqrt[3]{\frac{12B_U}{x^{(3)}(\ )}} \tag{5.123}$$

The first step is to find an expression for the third derivative:

$$x^{(3)}(\ ) \quad 3!\ {}_3x_{n+1} \tag{5.124}$$

$${}_3x_{n+1} = \frac{{}_2x_{n+1} - {}_2x_n}{t_{n+1} - t_{n-2}} \tag{5.125}$$

$$= \frac{{}_2x_{n+1} - {}_2x_n}{h_{n+1} + h_n + h_{n-1}}, \tag{5.126}$$

$$= \frac{1}{h_{n+1} + h_n + h_{n-1}}\left(\frac{1}{h_{n+1} + h_n}\left(\frac{x_{n+1} - x_n}{h_{n+1}} - \frac{x_n - x_{n-1}}{h_n}\right) - \frac{1}{h_n + h_{n-1}}\left(\frac{x_n - x_{n-1}}{h_n} - \frac{x_{n-1} - x_{n-2}}{h_{n-1}}\right)\right) \tag{5.127}$$

Substituting the value for B_U and the approximation for the third derivative into equation (5.123) yields the bound for h:

$$h_{n+1} \quad \sqrt[3]{\frac{1}{10}\left|\frac{2}{{}_3x_{n+1}}\right|} \tag{5.128}$$

where ${}_3x_{n+1}$ is given in equation (5.127).

5.7 Differential-Algebraic Equations

Many classes of systems can be generically written in the form of:

$$F(t, y(t), \dot{y}(t)) = 0 \tag{5.129}$$

where F and $y \in R^m$. In some cases, equation (5.129) can be rewritten as

$$F(t, x(t), \dot{x}(t), y(t)) = 0 \tag{5.130}$$

$$g(t, x(t), y(t)) = 0 \tag{5.131}$$

In this form, the system of equations is typically known as a system of di erential-algebraic equations, or DAEs [5]. This form of DAEs may be considered to be a system of di erential equations (equation (5.130)) constrained to an algebraic manifold (equation(5.131)). Often the set of equations (5.131) are invertible (i.e. $y(t)$ can be obtained from $y(t) = g^{-1}(t, x(t))$) and equation (5.130) can be rewritten as an ordinary di erential equation:

$$F\left(t, x(t), \dot{x}(t), g^{-1}(t, x(t))\right) = f\left(t, x(t), \dot{x}(t)\right) = 0 \tag{5.132}$$

Although the set of ODEs may be conceptually simpler to solve, there are usually several compelling reasons to leave the system in its original DAE form. Many DAE models are derived from physical problems in which each variable has individual characteristics and physical signi“cance. Converting the DAE into an ODE may result in a loss of physical information in the solution. Furthermore, it may be more computationally expensive to obtain the solution of the system of ODEs since inherent sparsity may have been lost. Solving the original DAE directly provides greater information regarding the behavior of the system.

A special case of DAEs is the semi-explicit DAE:

$$\dot{x} = f(x, y, t) \quad x \in R^n \tag{5.133}$$

$$0 = g(x, y, t) \quad y \in R^m \tag{5.134}$$

where y has the same dimension as g. DAE systems that can be written in semi-explicit form are often referred to as *index 1* DAE systems [5]. The “rst concerted e ort to solve semi-explicit DAEs was proposed in [15] and later re“ned in [16], and consisted of replacing $\dot{x}(t)$ by a k-step backwards di erence formula (BDF) approximation

$$\dot{x}(t) \qquad \frac{\nabla_n x_n}{h_n} = \frac{1}{h_n} \sum_{i=0}^{k} \alpha_i x_{n-i} \tag{5.135}$$

and then solving the resulting equations

$$\nabla_n x_n = h_n f\left(x_n, y_n, t_n\right) \tag{5.136}$$

$$0 = g\left(x_n, y_n, t_n\right) \tag{5.137}$$

for approximations to x_n and y_n.

Various other numerical integration techniques have been studied for application to DAE systems. Variable step size/“xed formula code has been proposed to solve the system of DAEs [44]. Speci“cally, a classic fourth-order Runge-Kutta method was used to solve for $\dot{x}$

Applying the trapezoidal algorithm to the transient stability equations yields the following system of equations:

$$\delta_1(n+1) = \delta_1(n) + \frac{h}{2}[\omega_1(n+1) - \omega_s + \omega_1(n) - \omega_s] \tag{5.153}$$

$$\omega_1(n+1) = \omega_1(n) + \frac{h}{2}[f_1(n+1) + f_1(n)] \tag{5.154}$$

$$\delta_2(n+1) = \delta_2(n) + \frac{h}{2}[\omega_2(n+1) - \omega_s + \omega_2(n) - \omega_s] \tag{5.155}$$

$$\omega_2(n+1) = \omega_2(n) + \frac{h}{2}[f_2(n+1) + f_2(n)] \tag{5.156}$$

$$\delta_3(n+1) = \delta_3(n) + \frac{h}{2}[\omega_3(n+1) - \omega_s + \omega_3(n) - \omega_s] \tag{5.157}$$

$$\omega_3(n+1) = \omega_3(n) + \frac{h}{2}[f_3(n+1) + f_3(n)] \tag{5.158}$$

where

$$f_i(n+1) = \frac{1}{M_i}\left[P_{m_i} - E_i^2 G_{ii} - E_i \sum_{j=i}^{n} E_j (B_{ij}\sin\delta_{ij}(n+1) + G_{ij}\cos\delta_{ij}(n+1))\right] \tag{5.159}$$

Since the transient stability equations are nonlinear and the trapezoidal method is an implicit method, they must be solved iteratively using the Newton-Raphson method at each time point. The iterative equations are

$$\left[I - \frac{h}{2}J(n+1)^k\right]\begin{bmatrix}\Delta\delta_1(n+1)^{k+1}\\ \Delta\omega_1(n+1)^{k+1}\\ \Delta\delta_2(n+1)^{k+1}\\ \Delta\omega_2(n+1)^{k+1}\\ \Delta\delta_3(n+1)^{k+1}\\ \Delta\omega_3(n+1)^{k+1}\end{bmatrix} = \begin{bmatrix}\Delta\delta_1^k(n+1)\\ \Delta\omega_1^k(n+1)\\ \Delta\delta_2^k(n+1)\\ \Delta\omega_2^k(n+1)\\ \Delta\delta_3^k(n+1)\\ \Delta\omega_3^k(n+1)\end{bmatrix} - \begin{bmatrix}\delta_1(n)\\ \omega_1(n)\\ \delta_2(n)\\ \omega_2(n)\\ \delta_3(n)\\ \omega_3(n)\end{bmatrix} - \frac{h}{2}\begin{bmatrix}\omega_1^k(n+1) + \omega_1(n) - 2\omega_s\\ f_1^k(n+1) + f_1(n)\\ \omega_2^k(n+1) + \omega_2(n) - 2\omega_s\\ f_2^k(n+1) + f_2(n)\\ \omega_1^k(n+1) + \omega_1(n) - 2\omega_s\\ f_3^k(n+1) + f_3(n)\end{bmatrix} \tag{5.160}$$

where

$$[J] = \begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\\ f_{1\delta_1} & 0 & f_{1\delta_2} & 0 & f_{1\delta_3} & 0\\ 0 & 0 & 0 & 1 & 0 & 0\\ f_{2\delta_1} & 0 & f_{2\delta_2} & 0 & f_{2\delta_3} & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ f_{3\delta_1} & 0 & f_{3\delta_2} & 0 & f_{3\delta_3} & 0\end{bmatrix} \tag{5.161}$$

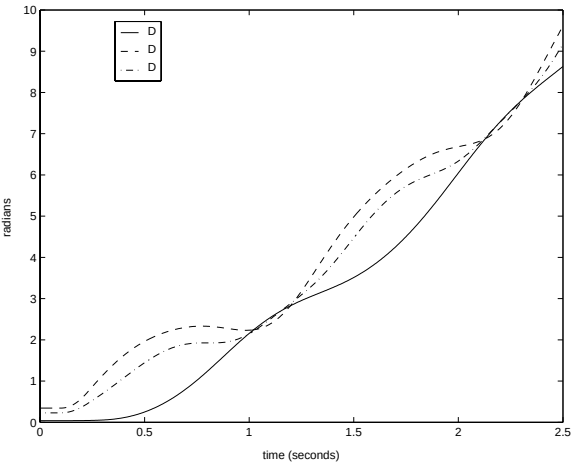


FIGURE 5.18
Rotor angle response for Example 5.9

Note that LU factorization must be employed to solve the discretized equations. These equations are iterated at each time point until convergence of the Newton-Raphson algorithm. The fault-on and post-fault matrices are substituted in at the appropriate times in the integration. The simulation results are shown in Figures 5.18 and 5.19 for the rotor angles and angular frequencies, respectively. From the waveforms shown in these “gures, it can be concluded that the system remains stable since the waveforms do not diverge during the simulation interval.

5.8.2 Mid-Term Stability Analysis

Reduction processes frequently destroy the natural physical structure and sparsity of the full order system. Numerical solution algorithms which make use of structure and sparsity for e cien cy perform poorly on the reduced-order system even though the reduced-order system is still quite large. After the “rst few seconds of a power system disturbance, the classical model representation may no longer be valid due to the dynamic behavior of the automatic voltage regulator, the turbine/governor system, under-load-tap-changing transformers, and the dynamic nature of some system loads. For mid-term stability analyses, a more detailed model is required to capture a wider range of system behavior. Since the behavior of loads may signi“cantly impact the stability of the system, it is desirable to be able to retain individual load buses during the simulation. This type of model is often referred to as a •structure-preservingŽ model, since the physical structure of the power system is retained. The inclusion of the load buses requires the solution of the set of power ”ow equations

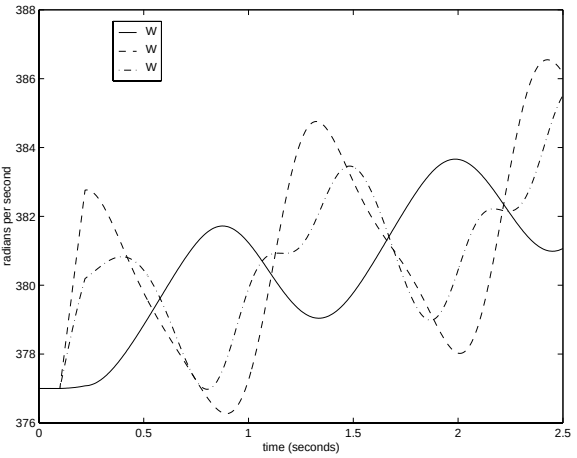


FIGURE 5.19
Angular frequency response for Example 5.9

governing the system network. This constraint leads to the inclusion of the algebraic power ”ow equations in conjunction with the di erential equations describing the states. One example of a structure-preserving DAE model is given below:

$$T_{d0_i} E_{q_i} = \check{S} E_{q_i} \check{S} \left(x_{d_i} \check{S} x_{d_i} \right) I_{d_i} + E_{fd_i} \tag{5.162}$$

$$T_{q0_i} E_{d_i} = \check{S} E_{d_i} + \left(x_{q_i} \check{S} x_{q_i} \right) I_{q_i} \tag{5.163}$$

$$_i = \ _i \check{S} \ _s \tag{5.164}$$

$$\frac{2H_i}{s} \ _i = T_{m_i} \check{S} E_{d_i} I_{d_i} \check{S} E_{q_i} I_{q_i} \check{S} \left(x_{q_i} \check{S} x_{d_i} \right) I_{d_i} I_{q_i} \tag{5.165}$$

$$T_{E_i} E_{fd_i} = \check{S} \left(K_{E_i} + S_{E_i} \left(E_{fd_i} \right) \right) E_{fd_i} + V_{R_i} \tag{5.166}$$

$$T_{F_i} R_{F_i} = \check{S} R_{F_i} + \frac{K_{F_i}}{T_{F_i}} E_{fd_i} \tag{5.167}$$

$$T_{A_i} V_{R_i} = \check{S} V_{R_i} + K_{A_i} R_{F_i} \check{S} \frac{K_{A_i} K_{F_i}}{T_{F_i}} E_{fd_i} + K_{A_i} \left(V_{ref_i} \check{S} V_{T_i} \right) \tag{5.168}$$

$$T_{RH_i} T_{M_i} = \check{S} T_{M_i} + \left(1 \check{S} \frac{K_{HP_i} T_{RH_i}}{T_{CH_i}} \right) P_{CH_i} + \frac{K_{HP_i} T_{RH_i}}{T_{CH_i}} P_{SV_i} \tag{5.169}$$

$$T_{CH_i} P_{CH_i} = \check{S} P_{CH_i} + P_{SV_i} \tag{5.170}$$

$$T_{SV_i} P_{SV_i} = \check{S} P_{SV_i} + P_{C_i} \check{S} \frac{1}{R} \ _i \tag{5.171}$$

and

$$0 = V_i e^{j\theta_i} + r_s + jx_{d_i}\,(I_{d_i} + jI_{q_i})\,e^{j(\theta_i - \delta_2)} \\ \delta\ E_{d_i} + x_{q_i}\,\delta\,x_{d_i}\ I_{q_i} + jE_{q_i}\ e^{j(\theta_i - \delta_2)} \tag{5.172}$$

$$0 = V_i e^{j\theta_i}\,(I_{d_i}\ \delta\ jI_{q_i})\,e^{\delta j(\theta_i - \delta_2)}\ \delta \sum_{k=1}^{N} V_i V_k Y_{ik}\, e^{j(\theta_i - \delta_k - \delta_{ik})} \tag{5.173}$$

$$0 = P_i + jQ_i\ \delta \sum_{k=1}^{N} V_i V_k Y_{ik}\, e^{j(\theta_i - \delta_k - \delta_{ik})} \tag{5.174}$$

These equations describe the behavior of a two-axis generator model, a simple automatic voltage regulator and exciter, a simple turbine/governor, and constant power loads. This set of dynamic equations is listed here for illustration purposes and is not intended to be inclusive of all possible representations. A detailed development of these equations may be found in [42].

These equations may be modeled in a more general form as

$$\dot{x} = f(x, y) \tag{5.175}$$

$$0 = g(x, y) \tag{5.176}$$

where the state vector x contains the dynamic state variables of the generators. The vector y is typically much larger and contains all of the network variables including bus voltage magnitude and angle. It may also contain generator states such as currents that are not inherently dynamic. There are two typical approaches to solving this set of differential/algebraic equations. The first is the method suggested by Gear whereby all equations are solved simultaneously. The second approach is to solve the differential and algebraic sets of equations separately and iterate between each.

Consider the first approach applied to the DAE system using the trapezoidal integration method:

$$x(n+1) = x(n) + \frac{h}{2}\left[f\,(x(n+1), y(n+1)) + f\,(x(n), y(n))\right] \tag{5.177}$$

$$0 = g(x(n+1), y(n+1)) \tag{5.178}$$

This set of nonlinear equations must then be solved using the Newton-Raphson method for the combined state vector $[x(n+1)\ \ y(n+1)]^T$:

$$\begin{bmatrix} I \delta \frac{h}{2}\frac{f}{x} & \delta \frac{h}{2}\frac{f}{y} \\ \frac{g}{x} & \frac{g}{y} \end{bmatrix} \begin{bmatrix} x(n+1)^{k+1} \delta x(n+1)^k \\ y(n+1)^{k+1} \delta y(n+1)^k \end{bmatrix} = \begin{bmatrix} x(n+1)^k \delta x(n) \delta \frac{h}{2} f\ x(n+1)^k, y(n+1)^k\ + f\ (x(n), y(n)) \\ g\ x(n+1)^k, y(n+1)^k \end{bmatrix} \tag{5.179}$$

The advantages of this method are that since the whole set of system equations is used, the system matrices are quite sparse and sparse solution techniques

can be used efficiently. Secondly, since the set of equations are solved simultaneously, the iterations are more likely to converge since the only iteration involved is that of the Newton-Raphson algorithm. Note, however, that in a system of ODEs, the left hand side matrix (the matrix to be factored in LU factorization) can be made to be diagonally dominant (and therefore well-conditioned) by decreasing the step size h. In DAE systems, however, the left hand side matrix may be ill conditioned under certain operating points where $\frac{g}{y}$ is ill conditioned causing difficulty in solving the system of equations. This situation may occur during the simulation of voltage collapse where the states encounter a bifurcation. The subject of bifurcation and voltage collapse is complex and outside the scope of this book; however, several excellent texts have been published that study these phenomena in great detail [25] [42] [55].

The second approach to solving the system of DAEs is to solve each of the subsystems independently and iteratively. The system of differential equations is solved "rst for $x(n+1)$ while holding $y(n+1)$ constant as an input. After $x(n+1)$ has been found, it is then used as an input to solve the algebraic system. The updated value of $y(n+1)$ is then substituted back into the differential equations, and $x(n+1)$ is recalculated. This back and forth process is repeated until the values $x(n+1)$ and $y(n+1)$ converge. The solution is then advanced to the next time point. The advantage of this method is simplicity in programming, since each subsystem is solved independently and the Jacobian elements $\frac{f}{y}$ and $\frac{g}{x}$ are not used. In some cases, this may speed up the computation, although more iterations may be required to reach convergence.

5.9 Problems

1. Determine a Taylor-series expansion for the solution of the equation

$$\dot{x} = x^2 \quad x(0) = 1$$

about the point $x = 0$ (a McClaurin series expansion). Use this approximation to compute x for $x = 0.2$ and $x = 1.2$. Compare with the exact solution and explain the results.

2. Use the following algorithms to solve the initial value problem

$$\dot{x}_1 = -2x_2 + 2t^2 \quad x_1(0) = -4$$
$$\dot{x}_2 = \frac{1}{2}x_1 + 2t \quad x_2(0) = 0$$

on the interval $0 \quad t \quad 5$ with a "xed integration step of 0.25 seconds.

(a) Backward Euler
(b) Forward Euler
(c) Trapezoidal Rule
(d) Fourth Order Runge Kutta

Compare the answers to the exact solution

$$x_1(t) = -4\cos t$$
$$x_2(t) = -2\sin t + t^2$$

3. Consider a simple ecosystem consisting of rabbits that have an in"nite food supply and foxes that prey upon the rabbits for their food. A classical mathematical •predator-preyŽ model due to Volterra describes this system by a pair of non-linear, "rst-order di erential equations:

$$\dot{r} = r + rf \quad r(0) = r_0$$
$$\dot{f} = f + rf \quad f(0) = f_0$$

where $r = r(t)$ is the number of rabbits, $f = f(t)$ is the number of foxes. When $= 0$, the two populations do not interact, and so the rabbits multiply and the foxes die o from starvation.

Investigate the behavior of this system for $= -1$, $= 0.01$, $= 0.25$, and $= -0.01$. Use the trapezoidal integration method with $h = 0.1$ and

$T = 50$, and the set of initial conditions $r_0 = 30 \pm 10$ and $f_0 = 80 \pm 10$. Plot (1) t vs. r and f, and (2) r vs. f for each case.

4. Consider the following linear multistep formula

$$x_{n+1} = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3} + h b_{-1} f(x_{n+1}, t_{n+1})$$

   (a) What are the number of steps in the formula?

   (b) What is the maximum order that will enable you to determine all the coefficients of the formula?

   (c) Assuming uniform step size h, find all the coefficients of the formula such that its order is the answer of part (b).

   (d) Is the formula implicit or explicit?

   (e) Assuming uniform step size h, find the expression for the Local Truncation error of the formula.

   (f) Is the formula absolutely stable?

   (g) Is the formula stiffly-stable?

5. Consider the following initial value problem:

$$\dot{x} = 100(\sin(t) - x), \quad x(0) = 0;$$

   The exact solution is:

$$x(t) = \frac{\sin(t) - 0.01\cos(t) + 0.01e^{-100t}}{1.0001}$$

   Solve the initial value problem with $h = 0.02$ s using the following integration methods. Plot each of the numerical solutions against the exact solution over the range $t \in [0, 3.0]$ seconds. Plot the global error for each method over the range $t \in [0, 3.0]$ seconds. Discuss your results.

   (a) Backward Euler

   (b) Forward Euler

   (c) Trapezoidal Rule

   (d) Fourth Order Runge Kutta

   (e) Repeat (a)-(d) with step size $h = 0.03$ s.

6. The following system of equations are known as the Lorenz equations:

$$\dot{x} = \sigma(y - x)$$
$$\dot{y} = \rho x - y - xz$$
$$\dot{z} = xy - \beta z$$

Let $\sigma = 10$, $\rho = 28$, and $\beta = 2.67$. Use the trapezoidal method to plot the response of the system for $0 \le t \le 10$ seconds using a fixed integration time step of $h = 0.005$ seconds and a Newton-Raphson convergence error of 10^{-5}. Plot x versus y in two dimensions and x vs. y vs. z in three dimensions.

(a) Use $[x(0)\ \ y(0)\ \ z(0)]^T = [20\ \ 20\ \ 20]^T$.

(b) Use $[x(0)\ \ y(0)\ \ z(0)]^T = [21\ \ 20\ \ 20]^T$. Explain the difference.

7. Consider the following system of stiff equations:

$$\dot{x}_1 = -2x_1 + x_2 + 100 \quad x_1(0) = 0$$
$$\dot{x}_2 = 10^4 x_1 - 10^4 x_2 + 50 \quad x_2(0) = 0$$

(a) Determine the maximum step size h_{max} for the forward Euler algorithm to remain numerically stable.

(b) Use the forward Euler algorithm with step size $h = \frac{1}{2} h_{max}$ to solve for $x_1(t)$ and $x_2(t)$ for $t > 0$.

(c) Repeat using a step size $h = 2\,h_{max}$.

(d) Use the backward Euler algorithm to solve the stiff equations. Choose the following step sizes in terms of the maximum step size h_{max}.

i. $h = 10\,h_{max}$
ii. $h = 100\,h_{max}$
iii. $h = 1000\,h_{max}$
iv. $h = 10,000\,h_{max}$

(e) Repeat using the multistep method (Gear's method) of Problem 4.

8. Consider a multistep method of the form

$$x_{n+2} - x_{n-2} + \alpha(x_{n+1} - x_{n-1}) = h[\beta(f_{n+1} + f_{n-1}) + \gamma f_n]$$

(a) Show that the parameters α, β, and γ can be chosen uniquely so that the method has order $p = 6$.

(b) Discuss the stability properties of this method. For what region is it absolutely stable? Stiffly stable?

9. A power system can be described by the following ODE system:

$$\dot{\delta}_i = \omega_i - \omega_s$$

$$M_i \dot{\omega}_i = P_i - E_i \sum_{k=1}^{n} E_k Y_{ik} \sin(\delta_i - \delta_k - \theta_{ik})$$

Using a step size of $h = 0.01$ s and the trapezoidal integration method, determine whether or not this system is stable for a clearing time of 4 cycles.

Let

$$E_1 = 1.0566 \angle 2.2717$$
$$E_2 = 1.0502 \angle 19.7315$$
$$E_3 = 1.0170 \angle 13.1752$$
$$P_1 = 0.716$$
$$P_2 = 1.630$$
$$P_3 = 0.850$$

and the following admittance matrices:

Prefault:

$$\begin{bmatrix} 0.846 - j\,2.988 & 0.287 + j\,1.513 & 0.210 + j\,1.226 \\ 0.287 + j\,1.513 & 0.420 - j\,2.724 & 0.213 + j\,1.088 \\ 0.210 + j\,1.226 & 0.213 + j\,1.088 & 0.277 - j\,2.368 \end{bmatrix}$$

Fault-on:

$$\begin{bmatrix} 0.657 - j\,3.816 & 0.000 + j\,0.000 & 0.070 + j\,0.631 \\ 0.000 + j\,0.000 & 0.000 - j\,5.486 & 0.000 + j\,0.000 \\ 0.070 + j\,0.631 & 0.000 + j\,0.000 & 0.174 - j\,2.796 \end{bmatrix}$$

Post-Fault:

$$\begin{bmatrix} 1.181 - j\,2.229 & 0.138 + j\,0.726 & 0.191 + j\,1.079 \\ 0.138 + j\,0.726 & 0.389 - j\,1.953 & 0.199 + j\,1.229 \\ 0.191 + j\,1.079 & 0.199 + j\,1.229 & 0.273 - j\,2.342 \end{bmatrix}$$

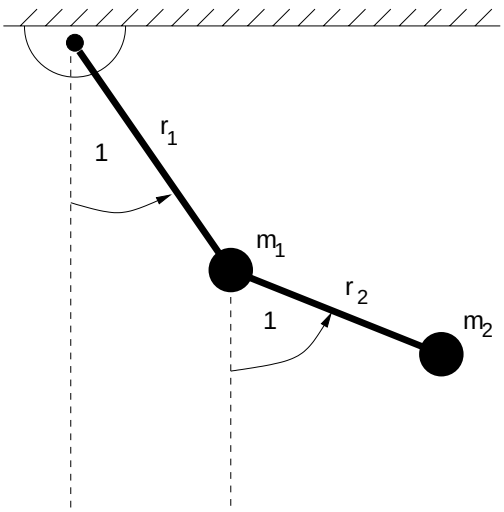


FIGURE 5.20
Double pendulum system

10. A double pendulum system is shown in Figure 5.20. Masses m_1 and m_2 are connected by massless rods of length r_1 and r_2. The equations of motion of the two masses, expressed in terms of the angles $_1$ and $_2$ as indicated, are:

$$\begin{aligned} -(m_1 + m_2)gr_1 \sin\ _1 &= (m_1 + m_2)r_1^2\ddot{}_1 + m_2 r_1 r_2\ddot{}_2 \cos(\ _1 - \ _2) \\ &\quad + m_2 r_1 r_2\ {}_2^2 \sin(\ _1 - \ _2) \\ -m_2 g r_2 \sin\ _2 &= m_2 r_2^2\ddot{}_2 + m_2 r_1 r_2\ddot{}_1 \cos(\ _1 - \ _2) \\ &\quad - m_2 r_1 r_2\ {}_1^2 \sin(\ _1 - \ _2) \end{aligned}$$

(a) Choose $x_1 = \ _1, x_2 = \ _1, x_3 = \ _2, x_4 = \ _2$, show that $[0; 0; 0; 0]$ is an equilibrium of the system.

(b) Show that the second-order Adams-Bashforth method is given by:

$$x_{n+1} = x_n + h\left[\frac{3}{2}f(x_n, t_n) - \frac{1}{2}f(x_{n-1}, t_{n-1})\right]$$

with

$$_T = \frac{5}{12}x^{(3)}(\ )h^3$$

(c) Show that the third-order Adams-Bashforth method is given by:

$$x_{n+1} = x_n + h\left[\frac{23}{12}f(x_n, t_n) - \frac{16}{12}f(x_{n-1}, t_{n-1}) + \frac{5}{12}f(x_{n-2}, t_{n-2})\right.$$

with

$$_T = \frac{3}{8} x^{(4)}(\ )h^4$$

(d) Let $r_1 = 1, r_2 = 0.5$, and $g = 10$. Using the second-order Adams-Bashforth method with h=0.005, plot the behavior of the system for an initial displacement of $_1 = 25$ and $_2 = 10$, for T [0, 10] with

i.

Thus the variance of e can be calculated from

$$ee^T = (z - \hat{z})(z - \hat{z})^T \quad (6.38)$$

$$= \left[I - A\left(A^T W A\right)^{-1} A^T W\right] ee^T \left[I - WA\left(A^T W A\right)^{-1} A^T\right] \quad (6.39)$$

The expected, or mean, value of ee^T is given by

$$E\left[ee^T\right] = \left[I - A\left(A^T W A\right)^{-1} A^T W\right] E\left[ee^T\right] \left[I - WA\left(A^T W A\right)^{-1} A^T\right] \quad (6.40)$$

Recall that $E\left[ee^T\right]$ is just the covariance matrix $R = W^{-1}$, which is a diagonal matrix. Thus

$$E\left[ee^T\right] = \left[I - A\left(A^T W A\right)^{-1} A^T W\right]\left[I - A\left(A^T W A\right)^{-1} A^T W\right] R \quad (6.41)$$

The matrix

$$\left[I - A\left(A^T W A\right)^{-1} A^T W\right]$$

has the unusual property that it is an idempotent matrix. An idempotent matrix M has the property that $M^2 = M$; thus no matter how many times M is multiplied by itself, it will still return the product M. Therefore,

$$E\left[ee^T\right] = \left[I - A\left(A^T W A\right)^{-1} A^T W\right]\left[I - A\left(A^T W A\right)^{-1} A^T W\right] R \quad (6.42)$$

$$= \left[I - A\left(A^T W A\right)^{-1} A^T W\right] R \quad (6.43)$$

$$= R - A\left(A^T W A\right)^{-1} A^T \quad (6.44)$$

$$= R \quad (6.45)$$

To determine whether the estimated values differ significantly from the measured values, a useful statistical measure is the χ^2 (chi-squared) test of inequality. This measure is based on the χ^2 probability distribution which differs in shape depending on its degrees of freedom, k, which is the difference between the number of measurements and the number of states. By comparing the weighted sum of errors with the χ^2 value for a particular degree of freedom and significance level, it can be determined whether the errors exceed the bounds of what would be expected by chance alone. A significance level indicates the level of probability that the measurements are erroneous. A significance level of 0.05 indicates there is a 5% likelihood that bad data exist, or conversely, a 95% level of confidence in the goodness of the data. For example, for $k = 2$ and a significance level $= 0.05$, if the weighted sum of errors does not exceed a χ^2 of 5.99, then the set of measurements can be assured of being good with 95% confidence; otherwise the data must be rejected as containing at least one bad data point. Although the χ^2 test is effective in signifying the presence of bad data, it cannot identify locations. The identification of bad data locations continues to be an open research topic.

^{2} Values

k	0.10	0.05	0.01	0.001
1	2.71	3.84	6.64	10.83
2	4.61	5.99	9.21	13.82
3	6.25	7.82	11.35	16.27
4	7.78	9.49	13.23	18.47
5	9.24	11.07	15.09	20.52
6	10.65	12.59	16.81	22.46
7	12.02	14.07	18.48	24.32
8	13.36	15.51	20.09	26.13
9	14.68	16.92	21.67	27.88
10	15.99	18.31	23.21	29.59
11	17.28	19.68	24.73	31.26
12	18.55	21.03	26.22	32.91
13	19.81	22.36	27.69	34.53
14	21.06	23.69	29.14	36.12
15	22.31	25.00	30.68	37.70
16	23.54	26.30	32.00	39.25
17	24.77	27.59	33.41	40.79
18	25.99	28.87	34.81	42.31
19	27.20	30.14	36.19	43.82
20	28.41	31.41	37.67	45.32
21	29.62	32.67	38.93	46.80
22	30.81	33.92	40.29	48.27
23	32.00	35.17	41.64	49.73
24	33.20	36.42	42.98	51.18
25	34.38	37.65	44.31	52.62
26	35.56	38.89	45.64	54.05
27	36.74	40.11	46.96	55.48
28	37.92	41.34	48.28	56.89
29	39.09	42.56	49.59	58.30
30	40.26	43.77	50.89	59.70

A test procedure to test for the existence of bad data is given by:

Test Procedure for Bad Data

1. Use z to estimate x
2. Calculate the error
$$e = z - Ax$$
3. Evaluate the weighted sum of squares
$$f = \sum_{i=1}^{m} \frac{1}{\sigma_i^2} e_i^2$$
4. For $k = m - n$ and a specified probability α, if $f < \chi^2_{k,\alpha}$ then the data are good; otherwise at least one bad data point exists.

Example 6.3
Using the chi-square test of inequality with $\alpha = 0.01$, check for the presence of bad data in the measurements of Example 6.1.

Solution 6.3 The number of states in Example 6.1 is 2 and the number of measurements is 4; therefore $k = 4 - 2 = 2$. The weighted sum of squares is given by

$$\begin{aligned} f &= \sum_{i=1}^{m=4} \frac{1}{\sigma_i^2} e_i^2 \\ &= 100(0.0141)^2 + 100(0.0108)^2 + 50(-0.0616)^2 + 50(0.0549) \\ &= 0.3720 \end{aligned}$$

From the table of chi-squared values, the chi-square value for this example is 9.21. The weighted least squares error is less than the chi-square value; thus, this indicates that the estimated values are good to a confidence level of 99%.

6.1.3 Nonlinear Least Squares State Estimation

As in the linear least squares estimation, the nonlinear least squares estimation attempts to minimize the square of the errors between a known set of measurements and a set of weighted nonlinear functions:

$$\text{minimize } f = \|e\|^2 = e^T \cdot e = \sum_{i=1}^{m} \frac{1}{2} [z_i - h_i(x)]^2 \qquad (6.46)$$

where $x \in R^n$ is the vector of unknowns to be estimated, $z \in R^m$ is the vector of measurements, σ_i^2 is the variance of the i^{th} measurement, and $h(x)$ is the

function vector relating x to z, where the measurement vector z can be a set of geographically distributed measurements, such as voltages and power flows.

In state estimation, the unknowns in the nonlinear equations are the state variables of the system. The state values that minimize the error are found by setting the derivatives of the error function to zero:

$$F(x) = H_x^T R^{-1}[z - h(x)] = 0 \tag{6.47}$$

where

$$H_x = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots & \frac{\partial h_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \frac{\partial h_m}{\partial x_2} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix} \tag{6.48}$$

and R is the matrix of measurement variances. Note that equation (6.47) is a set of nonlinear equations that must be solved using Newton-Raphson or another iterative numerical solver. In this case, the Jacobian of $F(x)$ is

$$J_F(x) = H_x^T(x)R^{-1}\Big|_x [z - h(x)] \tag{6.49}$$

$$= -H_x^T(x)R^{-1}H_x(x) \tag{6.50}$$

and the Newton-Raphson iteration becomes

$$H_x^T\left(x^k\right) R^{-1} H_x\left(x^k\right)\left(x^{k+1} - x^k\right) = H_x^T\left(x^k\right) R^{-1}\left(z - h(x^k)\right) \tag{6.51}$$

which is solved repeatedly using LU factorization. At convergence, x^{k+1} is equal to the set of states that minimize the error function f of equation (6.46). The test procedure for bad data is the same as that for the linear state estimation.

6.2 Linear Programming

Linear programming is one of the most successful forms of optimization. Linear programming can be used when a problem can be expressed by a linear objective (or cost) function to be maximized (or minimized) subject to linear equality or inequality constraints. A general linear programming problem can be formulated as

$$\text{minimize} \quad f(x) = c^T x \tag{6.52}$$

$$\text{subject to} \quad Ax \quad b \tag{6.53}$$

$$x \quad 0 \tag{6.54}$$

Note that almost any linear optimization problem can be put into this form via one of the following tranformations:

1. maximizing $c^T x$ is the same as minimizing $-c^T x$
2. any constraint of the form $a^T x$ is equivalent to $-a^T x$
3. any constraint of the form $a^T x =$ is equivalent to $a^T x$ and $-a^T x$
4. if a problem does not require x_i to be nonnegative, then x_i can be replaced by the difference of two variables $x_i = u_i - v_i$ where u_i and v_i are nonnegative.

Any linear programming problem described by (A, b, c), there exists another equivalent, or dual problem $(-A^T, -c, -b)$. If a linear programming problem and its dual both have feasible points (i.e. any point that satisfies $Ax \quad b$, $x \quad 0$ or $-A^T y \quad -c$, $y \quad 0$ for the dual problem), then both problems have solutions and their values are the negatives of each other.

6.2.1 Simplex Method

One of the most common methods of solving linear programming problems is the well-known simplex method. The simplex method is an iterative method that moves the x vector from one feasible basic vector to another in such a way that $f(x)$ always decreases. It gives the exact result after a number of steps which is usually much less than $n - m$, generally taking $2m$ to $3m$ iterations at most (where m is the number of equality constraints) [7]. However, its worst-case complexity is exponential, as can be demonstrated with carefully constructed examples.

The simplex method is often accomplished by representing the problem in tableau form, which is then modified in successive steps according to given rules. Every step of the simplex method begins with a tableau. The top row contains the coefficients that pertain to the objective function $f(x)$. The current value of $f(x)$ is displayed in the top right corner of the tableau. The next m rows in the tableau represent the equality constraints. The last row of the tableau contains the current x vector. The rows in the tableau pertaining to the equality constraints can be transformed by elementary row operations without altering the solution.

The rules of the tableau that must be satisfied are:

1. The x vector must satisfy the equality constraints $Ax = b$
2. The x vector must satisfy the inequality $x \quad 0$
3. There are n components of x (designated nonbasic variables) that are zero. The remaining m components are usually nonzero and are designated as basic variables
4. In the matrix that defines the constraints, each basic variable occurs in only one row.

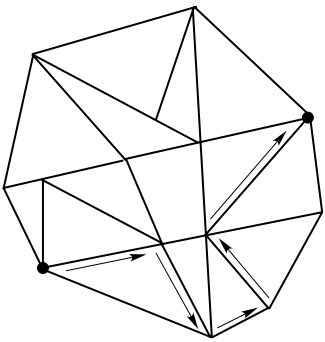


FIGURE 6.2
Example of simplex method search

5. The objective function f (x) must be expressed only in terms of nonbasic variables.

6. An arti“cial variable may be added to one or more constraints to obtain a starting solution.

The Simplex Algorithm is summarized:

Simplex Algorithm

1. If all coe cients in f (x) (i.e. top row of the tableau) are greater than or equal to zero, then the current x vector is the solution.

2. Select the nonbasic variable whose coe cient in f (x) is the largest negative entry. This variable becomes the new basic variable x_j .

3. Divide each b_i by the coe cient of the new basic variable in that row, a_{ij} . The value assigned to the new basic variable is the least of these ratios (i.e. $x_j = b_k / a_{kj}$).

4. Using pivot element a_{kj} , create zeros in column j of A with Gaussian elimination. Return to 1.

The series of inequalities in equation (6.53) when taken together form intersecting hyperplanes. The feasible region is the interior of this n-dimensional polytope and the minimum f (x) must occur on an edge or vertex of this polytope. The simplex method is an organized search of the vertices by moving along the steepest edge of the polytope until k is obtained as shown in Figure 6.2.

Example 6.4
Minimize

$$f(x): \quad -6x_1 - 14x_2$$

subject to the following constraints:

$$\begin{aligned} 2x_1 + x_2 &\le 12 \\ 2x_1 + 3x_2 &\le 15 \\ x_1 + 7x_2 &\le 21 \\ x_1 \ge 0,\ x_2 &\ge 0 \end{aligned}$$

Solution 6.4 Introduce slack variables x_3, x_4, and x_5 such that the problem becomes:

Minimize

$$f(x): \quad -6x_1 - 14x_2 + 0x_3 + 0x_4 + 0x_5$$

subject to the following constraints:

$$\begin{aligned} 2x_1 + x_2 + x_3 &= 12 \\ 2x_1 + 3x_2 + x_4 &= 15 \\ x_1 + 7x_2 + x_5 &= 21 \end{aligned}$$

$$x_1 \ge 0,\ x_2 \ge 0,\ x_3 \ge 0,\ x_4 \ge 0,\ x_5 \ge 0$$

Form the tableau:

−6	−14	0	0	0	0
2	1	1	0	0	12
2	3	0	1	0	15
1	7	0	0	1	21
0	0	12	15	21	

The starting vector is $x = [0\ 0\ 12\ 15\ 21]^T$. The current value of $f(x)$ is displayed in the top right corner. In the next step, the function $f(x)$ is examined to determine which variable will cause the greatest decrease. Since -14 is more negative than -6, a unit increase in x_2 will decrease $f(x)$ faster than a unit increase in x_1. Therefore, hold x_1 constant at zero and allow x_2 to increase as much as possible (i.e. traverse one edge of the polytope to the next vertex). To determine the new value of x_2 consider the constraints:

$$\begin{aligned} 0 &\le x_3 = 12 - x_2 \\ 0 &\le x_4 = 15 - 3x_2 \\ 0 &\le x_5 = 21 - 7x_2 \end{aligned}$$

From these constraints, the possible values of x_2 are $x_2 \leq 12$, $x_2 \leq 5$, $x_2 \leq 3$. The most stringent is $x_2 \leq 3$, therefore x_2 can be increased to 3, and

$$x_3 = 12 - x_2 = 9$$
$$x_4 = 15 - 3x_2 = 6$$
$$x_5 = 21 - 7x_2 = 0$$

yielding the new vector $x = [0\ 3\ 9\ 6\ 0]^T$ and $f(x) = -42$.

The new basic (non-zero) variables are x_2, x_3, and x_4, therefore $f(x)$ must be expressed in terms of x_1 and x_5. By substitution,

$$x_2 = \frac{(21 - x_5)}{7}$$

and

$$f(x) = -6x_1 - 14x_2 = -6x_1 - 14\frac{(21 - x_5)}{7} = -6x_1 + 2x_5 - 42$$

To satisfy the rule that a basic variable must occur in only one row, Gaussian elimination is used to eliminate x_2 from every row but one using the pivot as defined in Step 3 of the algorithm.

The new tableau after Gaussian elimination is:

$$\begin{bmatrix} -6 & 0 & 0 & 0 & -2 & -42 \\ \frac{13}{7} & 0 & 1 & 0 & -\frac{1}{7} & 9 \\ \frac{11}{7} & 0 & 0 & 1 & -\frac{3}{7} & 6 \\ 1 & 7 & 0 & 0 & 1 & 21 \\ 0 & 3 & 9 & 6 & 0 & \end{bmatrix}$$

The method is again repeated. Any increase in x_5 will increase $f(x)$, so x_1 is chosen to become a basic variable. Therefore x_5 is held at zero and x_1 is allowed to increase as much as possible. The new constraints are:

$$0 \leq x_3 = 9 - \frac{13}{7}x_1$$
$$0 \leq x_4 = 6 - \frac{11}{7}x_1$$
$$0 \leq 7x_2 = 21 - x_1$$

or $x_1 \leq \frac{63}{13}$, $x_1 \leq \frac{42}{11}$, and $x_1 \leq 21$. The most stringent constraint is $x_1 \leq \frac{42}{11}$, therefore x_1 is set to $\frac{42}{11}$ and the new values of x_2, x_3 and x_4 are computed. The new vector is $x = \left[\frac{42}{11}\ \frac{27}{11}\ \frac{21}{11}\ 0\ 0\right]^T$ and $f(x)$ is rewritten in terms of x_4 and x_5:

$$x_1 = \frac{7}{11}(6 - x_4)$$
$$f(x) = -6x_1 + 2x_5 - 42$$
$$= \frac{42}{11}x_4 + 2x_5 - \frac{714}{11}$$

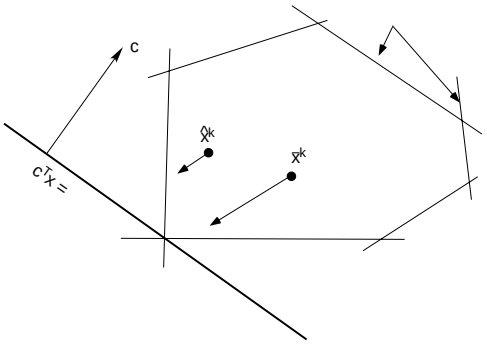


FIGURE 6.3
Comparison of two different points in the interior

Since all coefficients in $f(x)$ are positive, the simplex method terminates because any increase in x_4 or x_5 will increase $f(x)$. This signifies that the current x vector is the solution and the final value of $f(x)$ is $f\left(\frac{42}{11}, \frac{27}{11}\right) = \frac{630}{11}$.

6.2.2 Interior Point Method

A different type of method for linear programming problems are interior point methods (also known as Karmarkar's methods), whose complexity is polynomial for both average and worst case. The simplex method has the potential to have a worst case scenario of exponential complexity that can occur in the situation in which the solution visits every vertex in the feasible region before reaching the optimum. For this reason, interior point methods have received considerable attention over the past few decades. Interior point methods construct a sequence of strictly feasible points (i.e., lying in the interior of the polytope but never on its boundary) that converges to the solution.

The interior point method constructs a series of feasible solutions $x^0, x^1, \ldots$ that must satisfy $Ax_i = b$. Since x^0 satisfies $Ax^0 = b$ and the next point must satisfy $Ax^1 = b$, the difference in solutions must satisfy $A\ x = 0$. In other words, each step must lie in the nullspace of A, which is parallel to the feasible set. Projecting $-c$ onto that nullspace gives the direction of most rapid change. However, if the iterate x^k is close to the boundary (as with x^k in Figure 6.3), very little improvement will occur. If however, the current iterate is near the center (as with $\bar{x}^k$) there could be significant improvement. One of the key aspects of the interior point method is that a tranformation is applied such that the current feasible point is moved (through the transformation) to the center of the interior. The new direction is then computed and the interior point is transformed back to the original space.

This direction of change is the projected gradient direction, or p^k, and the

feasible points are updated through

$$x^{k+1} = x^k + \alpha p^k \tag{6.55}$$

where $\alpha > 0$ is the steplength. Since the feasible points must lie in the null space of A, each p^k must be orthogonal to the rows of A. The projection matrix P

$$P = I - A^T(AA^T)^{-1}A \tag{6.56}$$

will transform any vector v into $Pv = p$ and p will be in the null space of A because AP is the zero matrix.

Since projecting $-c$ onto the nullspace gives the direction of most rapid change, then to maintain feasibility a new iterate must satisfy $p^k = -Pc$. To remain in the interior of the space, the steplength α is chosen at each step to ensure feasibility of the nonnegativity constraints. To ensure that the updates remain in the interior of the feasible space, the steplength is chosen to be less than the full distance to the boundary, usually $0.5 \leq \alpha \leq 0.98$.

The last aspect is the transformation required to center the iterate in the feasible space. This is accomplished by a scaling, such that the iterate is equidistant from all constraint boundaries in the transformed feasible space. Therefore, after rescaling $x^k = e$, where $e = [1\ 1\ \ldots\ 1]^T$. Let $D = \text{diag}(x^k)$ be the diagonal matrix with each component of the current iterate x^k on the diagonals. This is accomplished by letting $x = Dx$ so that $x^k = e$. The new problem to be solved then becomes:

$$\text{minimize} \quad c^T x = z \tag{6.57}$$

$$\text{subject to} \quad Ax \quad b \tag{6.58}$$

$$x \quad 0 \tag{6.59}$$

where $c = Dc$ and $A = AD$. After scaling, the projection matrix becomes:

$$P = I - A^T(AA^T)^{-1}A \tag{6.60}$$

Hence, at each iteration k, the iterate x^k is rescaled to $x^k = e$ and the update is given by

$$x^{k+1} = e - \alpha Pc \tag{6.61}$$

and then the updated iterate is transformed back to the original space:

$$x^{k+1} = Dx^{k+1} \tag{6.62}$$

This process repeats until $x^{k+1} - x^k < \varepsilon$. This process is often referred to as the Primal Affine interior point method [8]. The steps of the method are summarized:

Primal A ne Interior Point Method

1. Let k = 0.
2. Let D=diag(x^k)
3. Compute A = AD, c = Dc
4. Compute P from equation (6.60)
5. Set p^k = Pc
6. Set = Š $\min_j p_j^k$. The factor is used to determine the maximum steplength that can be taken before exiting the feasible region.
7. Compute
$$x^{k+1} = e + p^k$$
8. Compute x^{k+1} = D x^{k+1}
9. If x^{k+1} Š x^k < , then done. Else set k = k + 1. Go to step 2.

Example 6.5
Repeat Example 6.4 using the Primal A ne Interior Point Method.

Solution 6.5 The problem is restated (with slack variables included) for convenience:

Minimize
$$f(x): \quad Š6x_1 Š 14x_2 + 0x_3 + 0x_4 + 0x_5 = z$$
subject to the following constraints:
$$\begin{aligned} 2x_1 + x_2 + x_3 &= 12 \\ 2x_1 + 3x_2 + x_4 &= 15 \\ x_1 + 7x_2 + x_5 &= 21 \end{aligned}$$
$$x_1 \quad 0, \; x_2 \quad 0, \; x_3 \quad 0, \; x_4 \quad 0, \; x_5 \quad 0$$

A feasible initial starting solution is
$$x^o = [1 \;\; 1 \;\; 9 \;\; 10 \;\; 13]$$
with $z^0 = c^T x^0 = Š20$. The “rst scaling matrix is
$$D = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 9 & & \\ & & & 10 & \\ & & & & 13 \end{bmatrix}$$

The rescaled matrix A and objective function vector c are computed as:

$$A = AD = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 2 & 3 & 0 & 1 & 0 \\ 1 & 7 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & 9 & & & \\ & & 10 & & \\ & & & 13 & \end{bmatrix} = \begin{bmatrix} 2 & 1 & 9 & 0 & 0 \\ 2 & 3 & 0 & 10 & 0 \\ 1 & 7 & 0 & 0 & 13 \end{bmatrix}$$

$$c = Dc = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 9 & \\ & & & 10 \\ & & & & 13 \end{bmatrix} \begin{bmatrix} -6 \\ -14 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -6 \\ -14 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The projection matrix P is:

$$P = I - A^T(AA^T)^{-1}A$$

$$= \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} - \begin{bmatrix} 2 & 2 & 1 \\ 1 & 3 & 7 \\ 9 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 13 \end{bmatrix} \begin{bmatrix} 2 & 1 & 9 & 0 & 0 \\ 2 & 3 & 0 & 10 & 0 \\ 1 & 7 & 0 & 0 & 13 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 2 & 1 \\ 1 & 3 & 7 \\ 9 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 13 \end{bmatrix}$$

$$= \begin{bmatrix} 0.9226 & -0.0836 & -0.1957 & -0.1595 & -0.0260 \\ -0.0836 & 0.7258 & -0.0621 & -0.2010 & -0.3844 \\ -0.1957 & -0.0621 & 0.0504 & 0.0578 & 0.0485 \\ -0.1595 & -0.2010 & 0.0578 & 0.0922 & 0.1205 \\ -0.0260 & -0.3844 & 0.0485 & 0.1205 & 0.2090 \end{bmatrix}$$

The projected gradient is

$$p^0 = -Pc = -\begin{bmatrix} 0.9226 & -0.0836 & -0.1957 & -0.1595 & -0.0260 \\ -0.0836 & 0.7258 & -0.0621 & -0.2010 & -0.3844 \\ -0.1957 & -0.0621 & 0.0504 & 0.0578 & 0.0485 \\ -0.1595 & -0.2010 & 0.0578 & 0.0922 & 0.1205 \\ -0.0260 & -0.3844 & 0.0485 & 0.1205 & 0.2090 \end{bmatrix} \begin{bmatrix} -6 \\ -14 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 4.3657 \\ 9.6600 \\ -2.0435 \\ -3.7711 \\ -5.5373 \end{bmatrix}$$

Calculate $= -\min_j p_j^0 = 5.5373$. Rescale the current iterate to $x^0 =$

$D^{-1}x^0 = e$ and move to x^1 in the transformed space with $\alpha = 0.9$:

$$x^1 = \begin{matrix}1\\1\\1\\1\\1\end{matrix} + p^0 = \begin{matrix}1.7096\\2.5701\\0.6679\\0.3871\\0.1000\end{matrix}$$

Transforming this point back to the original space:

$$x^1 = Dx^1 = \begin{matrix}1.7096\\2.5701\\6.0108\\3.8707\\1.3000\end{matrix}$$

and the updated cost function is $c^T x^1 = -46.2383$.

Performing one more iteration (and omitting the detailed text) yields:

$$A = \begin{matrix}3.4191 & 2.5701 & 6.0108 & 0 & 0\\ 3.4191 & 7.7102 & 0 & 3.8707 & 0\\ 1.7096 & 1.7990 & 4 & 0 & 1.3000\end{matrix}$$

$$c = \begin{matrix}-10.2574\\-35.9809\\0\\0\\0\end{matrix}$$

$$P = \begin{matrix}0.5688 & -0.0584 & -0.2986 & -0.3861 & 0.0606\\ -0.0584 & 0.0111 & 0.0285 & 0.0295 & -0.0766\\ -0.2986 & 0.0285 & 0.1577 & 0.2070 & -0.0017\\ -0.3861 & 0.0295 & 0.2070 & 0.2822 & 0.0991\\ 0.0606 & -0.0766 & -0.0017 & 0.0991 & 0.9803\end{matrix}$$

$$p^1 = \begin{matrix}3.7321\\-0.2004\\-2.0373\\-2.8975\\-2.1345\end{matrix}$$

$$x^1 = \begin{matrix}2.1592\\0.9378\\0.3672\\0.1000\\0.3370\end{matrix}$$

$$x^1 = \begin{matrix} 3.6914 \\ 2.4101 \\ 2.2072 \\ 0.3871 \\ 0.4381 \end{matrix}$$

and the updated cost function is $c^T x^1 = -55.8892$.

This continues until $x^{k+1} - x^k < \varepsilon$ at which time the solution is:

$$x = \begin{matrix} 3.8182 \\ 2.4545 \\ 1.9091 \\ 0.0000 \\ 0.0000 \end{matrix}$$

and the updated cost function is $c^T x^1 = -57.2727$ both of which are the same as the simplex method.

6.3 Nonlinear Programming

Continuous nonlinear optimization problems are typically of the following form:

$$\text{minimize} \quad f(x) \quad x \quad R^n \tag{6.63}$$

$$\text{subject to} \quad c_i(x) = 0, \quad i \tag{6.64}$$

$$h_i(x) \quad 0, \quad i \tag{6.65}$$

where $[c(x) \quad h(x)]$ is an m-vector of nonlinear constraint functions such that and are nonintersecting index sets. The function $f(x)$ is sometimes referred to as a "cost" function. It is assumed throughout that f, c, and

Iteration 3

With $u = 0.4661$, solving for x_1 and x_2 again yields two values for each with a corresponding cost function:

$$x_1 = 0.5921 \quad x_2 = -0.7278 \quad f = 1.6271$$
$$x_1 = -3.5921 \quad x_2 = 3.4565 \quad f = 14.1799$$

The first set of values again leads to the minimum cost function, so they are selected as the operating solution. The di erence in cost functions is

$$\left|C^{(2)} - C^{(3)}\right| = |1.6276 - 1.6271| = 0.0005$$

which is greater than the stopping criterion. Substituting these values into the gradient function yields $C = 0.0541$ and the new value of u becomes:

$$\begin{aligned} u^{(4)} &= u^{(3)} - \gamma \quad C \\ &= 0.4661 - (0.05)(0.0541) \\ &= 0.4634 \end{aligned}$$

Iteration 4

With $u = 0.4634$, solving for x_1 and x_2 again yields two values for each with a corresponding cost function:

$$x_1 = 0.5850 \quad x_2 = -0.7315 \quad f = 1.6270$$
$$x_1 = -3.5850 \quad x_2 = 3.4385 \quad f = 14.1370$$

The first set of values again leads to the minimum cost function, so they are selected as the operating solution. The di erence in cost functions is

$$\left|C^{(3)} - C^{(4)}\right| = |1.6271 - 1.6270| = 0.0001$$

which satisfies the stopping criterion. Thus, the values $x_1 = 0.5850, x_2 = -0.7315$, and $u = 0.4634$ yield the minimum cost function $f = 1.6270$.

6.3.3 Sequential Quadratic Programming Algorithm

Gradient descent techniques work well for small nonlinear systems, but become ine cient as the dimension of the search space grows. The nonlinear sequential quadratic programming (SQP) optimization method is computationally e cient and has been shown to exhibit superlinear convergence for convex search spaces [4]. The SQP method also attempts to solve the system

$$\text{minimize} \quad f(x) \quad x \quad \mathsf{R}^n \tag{6.105}$$
$$\text{subject to} \quad c_i(x) = 0, \quad i \quad \xi \tag{6.106}$$
$$h_i(x) \geq 0, \quad i \quad \Xi \tag{6.107}$$

As before, the usual approach to solving this optimization problem is to use Lagrangian multipliers and minimize the hybrid system:

$$L(x, \lambda) = f(x) + \lambda^T c(x) + \pi^T h(x) \quad i = 1, \ldots, m \tag{6.108}$$

The KKT conditions are

$$\begin{aligned} f(x) + C^T\lambda + H^T\pi &= 0 \\ c(x) &= 0 \\ h(x) + s &= 0 \\ \pi^T s &= 0 \\ \pi, s &\geq 0 \end{aligned}$$

where λ is a vector of Lagrange multipliers for the equality constraints, π is a vector of Lagrange multipliers for the inequality constraints, s is a vector of slack variables and

$$C = \frac{\partial c(x)}{\partial x} \tag{6.109}$$

$$H = \frac{\partial h(x)}{\partial x} \tag{6.110}$$

This nonlinear system can be solved for x, λ, π, and s using the Newton-Raphson method. Consider the case of only x and λ. Applying the Newton-Raphson method to solve for $y = [x \;\; \lambda]^T$ with

$$F(y) = 0 = \begin{bmatrix} f(x) + C^T\lambda \\ c(x) \end{bmatrix}$$

and the Newton-Raphson update becomes

$$y^{k+1} = y^k - \alpha_k [\;\; F_k]^{-1} F(y^k)$$

or in original variables

$$\begin{bmatrix} x^{k+1} \\ \lambda^{k+1} \end{bmatrix} = \begin{bmatrix} x^k \\ \lambda^k \end{bmatrix} - \alpha_k \begin{bmatrix} {}^2L & c^T \\ c^T & 0 \end{bmatrix}_k^{-1} \begin{bmatrix} L \\ c \end{bmatrix}_k \tag{6.111}$$

where α_k is a positive steplength usually taken to be less than or equal to one.

Example 6.9

Repeat Example 6.8 using the SQP method.

Solution 6.9 The problem is repeated here:

Minimize

$$C: \quad x_1^2 + 2x_2^2 + u^2 = f(x_1, x_2, u) \tag{6.112}$$

subject to the following constraints:

$$0 = x_1^2 - 3x_2 + u - 3 \tag{6.113}$$
$$0 = x_1 + x_2 - 4u + 2 \tag{6.114}$$

Applying the KKT conditions yields the following nonlinear system of equations:

$$\begin{aligned} 0 &= 2x_1 + 2\lambda_1 x_1 + \lambda_2 \\ 0 &= 4x_2 - 3\lambda_1 + \lambda_2 \\ 0 &= 2u + \lambda_1 - 4\lambda_2 \\ 0 &= x_1^2 - 3x_2 + u - 3 \\ 0 &= x_1 + x_2 - 4u + 2 \end{aligned}$$

The Newton-Raphson iteration becomes

$$\begin{bmatrix} x_1 \\ x_2 \\ u \\ \lambda_1 \\ \lambda_2 \end{bmatrix}^{k+1} = \begin{bmatrix} x_1 \\ x_2 \\ u \\ \lambda_1 \\ \lambda_2 \end{bmatrix}^{k} - \begin{bmatrix} 2+2\lambda_1^k & 0 & 0 & 2x_1^k & 1 \\ 0 & 4 & 0 & -3 & 1 \\ 0 & 0 & 2 & 1 & -4 \\ 2x_1^k & -3 & 1 & 0 & 0 \\ 1 & 1 & -4 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2x_1^k + 2\lambda_1^k x_1^k + \lambda_2^k \\ 4x_2^k - 3\lambda_1^k + \lambda_2^k \\ 2u^k + \lambda_1^k - 4\lambda_2^k \\ (x_1^k)^2 - 3x_2^k + u^k - 3 \\ x_1^k + x_2^k - 4u^k + 2 \end{bmatrix} \tag{6.115}$$

Starting with an intitial guess: $\begin{bmatrix} x_1 & x_2 & u & \lambda_1 & \lambda_2 \end{bmatrix}^T = [1\ 1\ 1\ 1\ 1]^T$ yields the following updates:

k	x_1	x_2	u	λ_1	λ_2	$f(x)$
0	1.0000	1.0000	1.0000	1.0000	1.0000	4.0000
1	0.5774	-0.7354	0.4605	-0.9859	-0.0162	1.6270
2	0.8462	-0.5804	0.5664	-0.7413	0.0979	1.7106
3	0.6508	-0.7098	0.4853	-0.9442	0.0066	1.6666
4	0.5838	-0.7337	0.4625	-0.9831	-0.0145	1.6314
5	0.5774	-0.7354	0.4605	-0.9859	-0.0162	1.6270

which is the same solution as previously.

6.4 Power System Applications

6.4.1 Optimal Power Flow

Many power system applications, such as the power flow, o er only a snapshot of the system operation. Frequently, the system planner or operator is interested in the e ect that making adjustments to the system parameters will have on the power flow through lines or system losses. Rather than making the adjustments in a random fashion, the system planner will attempt to optimize the adjustments according to some objective function. This objective function can be chosen to minimize generating costs, reservoir water levels, or system losses, among others. The optimal power flow problem is to formulate the power flow problem to find system voltages and generated powers within the framework of the objective function. In this application, the inputs to the power flow are systematically adjusted to maximize (or minimize) a scalar function of the power flow state variables. The two most common objective functions are minimization of generating costs and minimization of active power losses.

The time frame of optimal power flow is on the order of minutes to one hour; therefore it is assumed that the optimization occurs using only those units that are currently on-line. The problem of determining whether or not to engage a unit, at what time, and for how long is part of the *unit commitment* problem and is not covered here. The minimization of active transmission losses saves both generating costs and creates a higher generating reserve margin.

Usually generator cost curves (the curves that relate generated power to the cost of such generation) are given as piecewise linear incremental costs curves. This has its origin in the simplification of concave cost functions with the valve points as cost curve breakpoints [19]. Piecewise linear incremental cost curves correspond to piecewise quadratic cost curves by integrating the incremental cost curves. This type of objective function lends itself easily to the *economic dispatch*, or λ-dispatch problem where only generating units are considered in the optimization. In this process, system losses and constraints on voltages and line powers are neglected. This economic dispatch method is illustrated in the following example.

Example 6.10

Three generators with the following cost functions serve a load of 952 MW. Assuming a lossless system, calculate the optimal generation scheduling.

$$\begin{aligned} C_1 &: \ P_1 + 0.0625P_1^2 \quad \$/hr \\ C_2 &: \ P_2 + 0.0125P_2^2 \quad \$/hr \\ C_3 &: \ P_3 + 0.0250P_3^2 \quad \$/hr \end{aligned}$$

Solution 6.10 The first step in determining the optimal scheduling of the generators is to construct the problem in the general form. Thus the optimization statement is:

$$\begin{aligned} \text{Minimize C:} \quad & P_1 + 0.0625P_1^2 + P_2 + 0.0125P_2^2 + P_3 + 0.0250P_3^2 \\ \text{Subject to:} \quad & P_1 + P_2 + P_3 - 952 = 0 \end{aligned}$$

From this statement, the constrained cost function becomes

$$C^*: \quad P_1 + 0.0625P_1^2 + P_2 + 0.0125P_2^2 + P_3 + 0.0250P_3^2 - \lambda\,(P_1 + P_2 + P_3 - 952) \tag{6.116}$$

Setting the derivatives of C^* to zero yields the following set of linear equations:

$$\begin{bmatrix} 0.125 & 0 & 0 & -1 \\ 0 & 0.025 & 0 & -1 \\ 0 & 0 & 0.050 & -1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ \lambda \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 952 \end{bmatrix} \tag{6.117}$$

Solving equation (6.117) yields

$$\begin{aligned} P_1 &= 112\ MW \\ P_2 &= 560\ MW \\ P_3 &= 280\ MW \\ \lambda &= \$15/MWhr \end{aligned}$$

for a constrained cost of \$7,616/hr.

This is the generation scheduling that minimizes the hourly cost of production. The value of λ is the *incremental* or *break-even* cost of production. This gives a company a price cut-o for buying or selling generation: if they can purchase generation for less than λ, then their overall costs will decrease. Likewise, if generation can be sold for greater than λ, their overall costs will decrease. Also note that at the optimal scheduling:

$$\lambda = 1 + 0.125P_1 = 1 + 0.025P_2 = 1 + 0.050P_3 \tag{6.118}$$

Since λ is the incremental cost for the system, this point is also called the point of "equal incremental cost," and the generation schedule is said to satisfy the "equal incremental cost criterion." Any deviation in generation from the equal increment cost scheduling will result in an increase in the production cost C.

Example 6.11

If a buyer is willing to pay \$16/MW hr for generation, how much excess generation should be produced and sold, and what is the profit for this transaction?

Solution 6.11 From Example 6.10, the derivatives of the augmented cost function yield the following relationships between generation and λ:

$$P_1 = 8(\lambda - 1)$$
$$P_2 = 40(\lambda - 1)$$
$$P_3 = 20(\lambda - 1)$$

from which the equality constraint yields:

$$8(\lambda - 1) + 40(\lambda - 1) + 20(\lambda - 1) - 952 = 0 \qquad (6.119)$$

To determine the excess amount, the equality equation (6.119) will be augmented and then evaluated at λ= \$16/MW hr:

$$8(16 - 1) + 40(16 - 1) + 20(16 - 1) - (952 + \text{excess}) = 0 \qquad (6.120)$$

Solving equation (6.120) yields a required excess of 68 MW, and P_1 = 120 MW, P_2 = 600 MW, and P_3 = 300 MW. The total cost of generation becomes

$$C: \quad P_1 + 0.0625P_1^2 + P_2 + 0.0125P_2^2 + P_3 + 0.0250P_3^2 = \$8,670/hr \qquad (6.121)$$

but the amount recovered by the sale of generation is the amount of excess times the incremental cost λ,

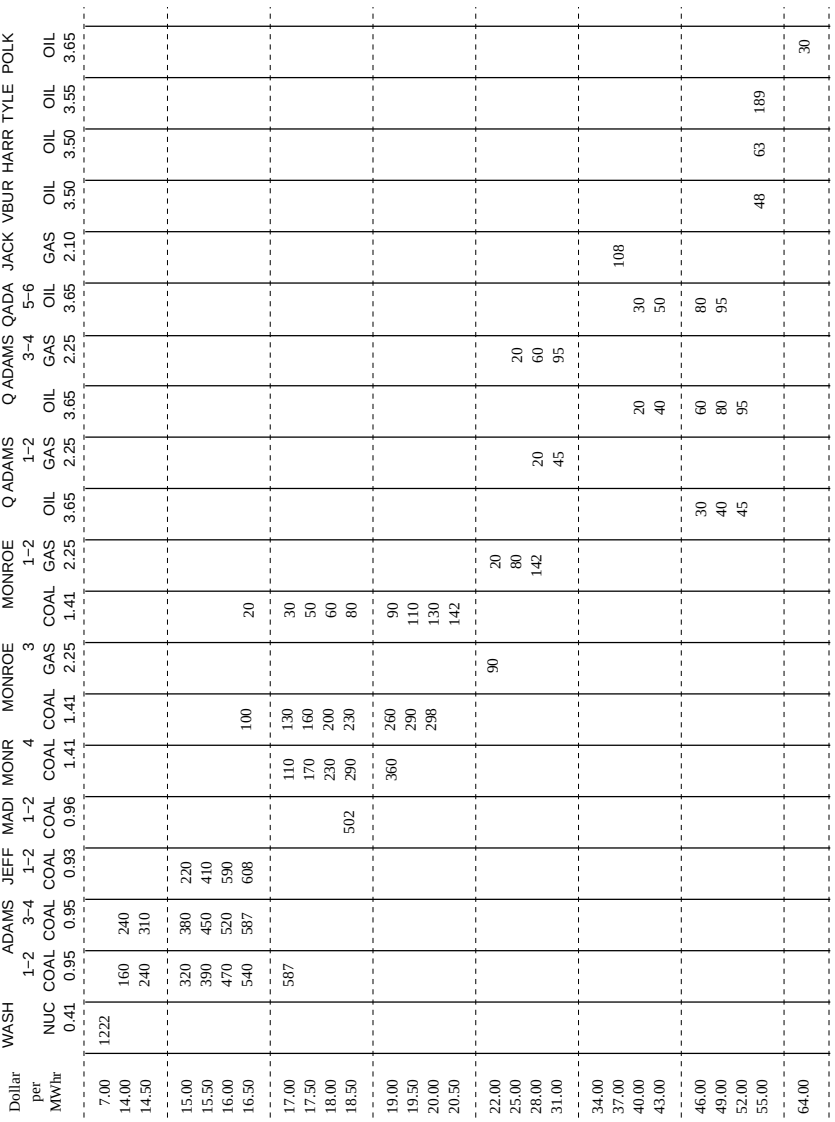
$$68MW \times \$16/MWhr = \$1,088/hr$$

Therefore, the total cost is \$8,670–1,088 = \$7,580/hr. This amount is \$34/hr less than the original cost of \$7,616/hr; thus, \$34/hr is the profit achieved from the sale of the excess generation at \$16/MW hr.

Figure 6.5 shows an incremental cost table for a medium size utility. The incremental cost of generation is listed vertically along the left hand side of the table. The various generating units are listed across the top from least expensive to most expensive (left to right). Nuclear units are among the least expensive units to operate and the nuclear unit *Washington* at the far left can produce up to 1222 MW at an incremental cost of 7.00 \$/MW hr. This incremental cost is half of the next least expensive unit *Adams* at 14 \$/MW hr which is a coal unit. As the available units become increasingly more expensive to operate, the incremental cost also increases.

Example 6.12
What is the incremental cost for the utility to produce 4500 MW?

Solution 6.12 To find the incremental cost that corresponds to 4500 MW from the Incremental Cost Table in Figure 6.5, the maximum generation available from each unit is summed until it equals 4500 MW. This amount is



Dollar per MWhr	WASH NUC 0.41	ADAMS 1–2 COAL 0.95	ADAMS 3–4 COAL 0.95	JEFF 1–2 COAL 0.93	MADI 1–2 COAL 0.96	MONR 4 COAL 1.41	MONROE 3 GAS 2.25	MONROE COAL 1.41	MONROE 1–2 GAS 2.25	Q ADAMS 1–2 OIL 3.65	Q ADAMS 1–2 GAS 2.25	Q ADAMS 3–4 OIL 3.65	Q ADAMS 3–4 GAS 2.25	QADA 5–6 OIL 3.65	JACK GAS 2.10	VBUR OIL 3.50	HARR OIL 3.50	TYLE OIL 3.55	POLK OIL 3.65
7.00	1222																		
14.00		160	240																
14.50		240	310																
15.00		320	380	220															
15.50		390	450	410															
16.00		470	520	590															
16.50		540	587	608		100		20											
17.00		587				110		30											
17.50						170		50											
18.00						230		60											
18.50					502	290		80											
19.00						360		90											
19.50								110											
20.00								130											
20.50								142											
22.00							90		20										
25.00									80				20						
28.00									142		20		60						
31.00											45		95						
34.00															108				
37.00												20		30					
40.00												40		50					
43.00																			
46.00										30		60		80					
49.00										40		80		95					
52.00										45		95							
55.00																48	63	189	
64.00																			30

FIGURE 6.5
Incremental cost table

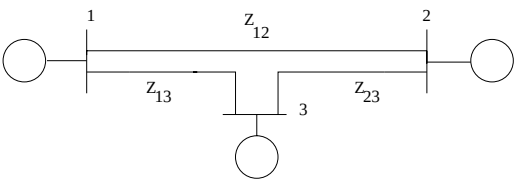


FIGURE 6.6
Figure for Example 6.13

reached by the gas unit of *Monroe 1-2*. This corresponds to an incremental cost of 28.00 $/MW hr. This is the breakeven point for 4500 MW.

If power can be purchased for less than 28.00 $/MW hr, the utility should purchase generation.

The primary drawback with the equal incremental cost scheduling is that it neglects all losses in the system. The only enforced equality constraint is that the sum of the generation must equal the total load demand. In reality, however, the sum of the generation must equal the load demand plus any system losses. In the consideration of system losses, the equality constraints must include the set of power flow equations, and the optimization process must to be extended to the steepest descent, or similar, approach [10].

Example 6.13

Consider the three machine system shown in Figure 6.6. This system has the same parameters as the three bus system of Example 3.9 except that bus 3 has been converted to a generator bus with a voltage magnitude of 1.0 pu. The total load and cost functions of the generators are the same as in Example 6.10. Using the equal cost criterion scheduling as a starting point, find the optimal scheduling of this system considering losses.

Solution 6.13 Following the steepest descent procedure detailed in Section 6.3.2, the first step is to develop an expression for the gradient C, where

$$C = \begin{bmatrix} \partial f \\ \partial u \end{bmatrix} - \begin{bmatrix} \partial g \\ \partial u \end{bmatrix}^T \left[\begin{bmatrix} \partial g \\ \partial x \end{bmatrix}^T \right]^{-1} \begin{bmatrix} \partial f \\ \partial x \end{bmatrix} \tag{6.122}$$

where f is the sum of the generator costs:

$$f: \quad C_1 + C_2 + C_3 = P_1 + 0.0625P_1^2 + P_2 + 0.0125P_2^2 + P_3 + 0.0250P_3^2$$

g is the set of load flow equations:

$$g_1: \quad 0 = P_2 - P_{L2} - V_2 \sum_{i=1}^{3} V_i Y_{2i} \cos(\delta_2 - \delta_i - \phi_{2i})$$

$$g_2: \quad 0 = P_3 - P_{L3} - V_3 \sum_{i=1}^{3} V_i Y_{3i} \cos(\delta_3 - \delta_i - \phi_{3i})$$

where P_{Li} denotes the active power load at bus i, the set of inputs u is the set of independent generation settings:

$$u = \begin{bmatrix} P_2 \\ P_3 \end{bmatrix}$$

and x is the set of unknown states

$$x = \begin{bmatrix} \delta_2 \\ \delta_3 \end{bmatrix}$$

The generator setting P_1 is not an input because it is the slack bus generation and cannot be independently set. From these designations, the various partial derivatives required for C can be derived:

$$\begin{bmatrix} \partial g \\ \partial u \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.123}$$

$$\begin{bmatrix} \partial g \\ \partial x \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial \delta_2} & \frac{\partial g_1}{\partial \delta_3} \\ \frac{\partial g_2}{\partial \delta_2} & \frac{\partial g_2}{\partial \delta_3} \end{bmatrix} \tag{6.124}$$

where

$$\frac{\partial g_1}{\partial \delta_2} = V_2 (V_1 Y_{12} \sin(\delta_2 - \delta_1 - \phi_{21}) + V_3 Y_{13} \sin(\delta_2 - \delta_3 - \phi_{23})) \tag{6.125}$$

$$\frac{\partial g_1}{\partial \delta_3} = -V_2 V_3 Y_{32} \sin(\delta_2 - \delta_3 - \phi_{23}) \tag{6.126}$$

$$\frac{\partial g_2}{\partial \delta_2} = -V_3 V_2 Y_{23} \sin(\delta_3 - \delta_2 - \phi_{32}) \tag{6.127}$$

$$\frac{\partial g_2}{\partial \delta_3} = V_3 (V_1 Y_{13} \sin(\delta_3 - \delta_1 - \phi_{31}) + V_2 Y_{23} \sin(\delta_3 - \delta_2 - \phi_{32})) \tag{6.128}$$

and

$$\begin{bmatrix} \partial f \\ \partial u \end{bmatrix} = \begin{bmatrix} 1 + 0.025 P_2 \\ 1 + 0.050 P_3 \end{bmatrix} \tag{6.129}$$

Finding the partial derivative $\begin{bmatrix} \partial f \\ \partial x \end{bmatrix}$ is slightly more difficult since the cost function is not written as a direct function of x. Recall, however, that P_1 is not an input, but is actually a quantity that depends on x, i.e.,

$$P_1 = V_1 (V_1 Y_{11} \cos(\delta_1 - \delta_1 - \phi_{11}) + V_2 Y_{12} \cos(\delta_1 - \delta_2 - \phi_{12}) + V_3 Y_{13} \cos(\delta_1 - \delta_3 - \phi_{13})) \tag{6.130}$$

Thus, using the chain rule,

$$\begin{bmatrix} \partial f \\ \partial x \end{bmatrix} = \begin{bmatrix} \partial f \\ \partial P_1 \end{bmatrix} \begin{bmatrix} \partial P_1 \\ \partial x \end{bmatrix} \tag{6.131}$$

$$= (1 + 0.125P_1) \begin{bmatrix} V_1 V_2 Y_{12} \sin(\delta_1 - \delta_2 - \phi_{12}) \\ V_1 V_3 Y_{13} \sin(\delta_1 - \delta_3 - \phi_{13}) \end{bmatrix} \tag{6.132}$$

From the previous example, the initial values of $P_2 = 0.56$ pu and $P_3 = 0.28$ pu are obtained from the equal incremental cost rule. Using $P_2 = 0.56$ pu and $P_3 = 0.28$ pu as inputs into the power flow yields the following states: $[\delta_2 \quad \delta_3] = [0.0286 \quad -0.0185]$ in radians and $P_1 = 0.1152$. Converting the generated powers to MW and substituting these values into the partial derivatives yields:

$$\begin{bmatrix} \partial g \\ \partial u \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.133}$$

$$\begin{bmatrix} \partial g \\ \partial x \end{bmatrix} = \begin{bmatrix} -13.3267 & 9.9366 \\ 9.8434 & -19.9219 \end{bmatrix} \tag{6.134}$$

$$\begin{bmatrix} \partial f \\ \partial u \end{bmatrix} = \begin{bmatrix} 15.0000 \\ 15.0000 \end{bmatrix} \tag{6.135}$$

$$\begin{bmatrix} \partial f \\ \partial x \end{bmatrix} = 15.4018 \begin{bmatrix} -52.0136 \\ -155.8040 \end{bmatrix} \tag{6.136}$$

which yields

$$C = \begin{bmatrix} -0.3256 \\ -0.4648 \end{bmatrix} \tag{6.137}$$

Thus, the new values for the input generation are:

$$\begin{bmatrix} P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 560 \\ 280 \end{bmatrix} - \gamma \begin{bmatrix} -0.3256 \\ -0.4648 \end{bmatrix} \tag{6.138}$$

With $\gamma = 1$, the updated generation is $P_2 = 560.3$ and $P_3 = 280.5$ MW.

Already the gradient C is very small, indicating that the generation values from the equal incremental cost process were relatively close to the optimal values, even considering losses. Proceeding with one more iteration yields the final generation values for all of the generators:

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 112.6 \\ 560.0 \\ 282.7 \end{bmatrix} \quad \text{MW}$$

which yields a cost of \$7,664/MW hr. Note that this amount is greater than the calculated cost for the equal incremental cost function. This increase is due to the extra generation required to satisfy the losses in the system.

Often the steepest descent method may indicate either states or inputs lie outside of their physical constraints. For example, the algorithm may result

in a power generation value that exceeds the physical maximum output of the generating unit. Similarly, the resulting bus voltages may lie outside of the desired range (usually ± 10% of unity). These are violations of the *inequality constraints* of the problem. In these cases, the steepest descent algorithm must be modified to reflect these physical limitations. There are several approaches to account for limitations and these approaches depend on whether or not the limitation is on the input (independent) or on the state (dependent).

6.4.1.1 Limitations on Independent Variables

If the application of the steepest descent algorithm results in an updated value of input that exceeds the specified limit, then the most straightforward method of handling this violation is simply to set the input state equal to its limit and continue with the algorithm except with one less degree of freedom.

Example 6.14

Repeat Example 6.13 except that the generators must satisfy the following limitations:

$$80 \le P_1 \le 1200 \text{ MW}$$
$$450 \le P_2 \le 750 \text{ MW}$$
$$150 \le P_3 \le 250 \text{ MW}$$

Solution 6.14 From the solution of Example 6.13, the output of generator 3 exceeds the maximum limit of 0.25 pu. Therefore after the first iteration in the previous example, P_3 is set to 0.25 pu. The new partial derivatives become:

$$\begin{bmatrix} \partial g \\ \partial u \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{6.139}$$

$$\begin{bmatrix} \partial g \\ \partial x \end{bmatrix} = \text{same} \tag{6.140}$$

$$\begin{bmatrix} \partial f \\ \partial u \end{bmatrix} = [1 + 0.025P_2] \tag{6.141}$$

$$\begin{bmatrix} \partial f \\ \partial x \end{bmatrix} = \text{same} \tag{6.142}$$

From the constrained steepest descent, the new values of generation become:

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 117.1 \\ 588.3 \\ 250.0 \end{bmatrix} \text{ MW}$$

with a cost of \$7,703/MW hr which is higher than the unconstrained cost of generation of \$7,664/MW hr. As more constraints are added to the system, the system is moved away from the optimal operating point, increasing the cost of generation.

6.4.1.2 Limitations on Dependent Variables

In many cases, the physical limitations of the system are imposed upon states that are dependent variables in the system description. In this case, the inequality equations are functions of x and must be added to the cost function. Examples of limitations on dependent variables include maximum line flows or bus voltage levels. In these cases, the value of the states cannot be independently set, but must be enforced indirectly. One method of enforcing an inequality constraint is to introduce a *penalty function* into the cost function. A penalty function is a function that is small when the state is far away from its limit, but becomes increasingly larger the closer the state is to its limit. Typical penalty functions include:

$$p(h) = e^{kh} \quad k > 0 \tag{6.143}$$

$$p(h) = x^{2n} e^{kh} \quad n, k > 0 \tag{6.144}$$

$$p(h) = a x^{2n} e^{kh} + b e^{kh} \quad n, k, a, b > 0 \tag{6.145}$$

and the cost function becomes

$$C^* : \quad C(u,x) + \lambda^T g(u,x) + p\,(h(u,x) - h^{max}) \tag{6.146}$$

This cost equation is then minimized in the usual fashion by setting the appropriate derivatives to zero. This method has the advantage of simplicity of implementation, but also has several disadvantages. The first disadvantage is that the choice of penalty function is often a heuristic choice and can vary by application. A second disadvantage is that this method cannot enforce *hard* limitations on states, i.e., the cost function becomes large if the maximum is exceeded, but the state is allowed to exceed its maximum. In many applications this is not a serious disadvantage. If the power flow on a transmission line slightly exceeds its maximum, it is reasonable to assume that the power system will continue to operate, at least for a finite length of time. If, however, the physical limit is the height above ground for an airplane, then even a slightly negative altitude will have dire consequences. Thus the use of penalty functions to enforce limits must be used with caution and is not applicable for all systems.

Example 6.15

Repeat Example 6.13, except use penalty functions to limit the power flow across line 2-3 to 0.4 per unit.

Solution 6.15 The power flow across line 2-3 in Example 6.13 is given by

$$P_{23} = V_2V_3Y_{23}\cos(\delta_2 - \delta_3 - \phi_{23}) - V_2^2Y_{23}\cos\phi_{23} \quad (6.147)$$
$$= 0.467 \text{ per unit}$$

If P_{23} exceeds 0.4 per unit, then the penalty function

$$p(h) = \left(1000V_2V_3Y_{23}\cos(\delta_2 - \delta_3 - \phi_{23}) - 1000V_2^2Y_{23}\cos\phi_{23} - 400\right)^2 \quad (6.148)$$

will be appended to the cost function. The partial derivatives remain the same with the exception of $\left[\frac{\partial f}{\partial x}\right]$ which becomes:

$$\begin{bmatrix}\frac{\partial f}{\partial x}\end{bmatrix} = \begin{bmatrix}\frac{\partial f}{\partial P_1}\end{bmatrix}\begin{bmatrix}\frac{\partial P_1}{\partial x}\end{bmatrix} + \begin{bmatrix}\frac{\partial f}{\partial P_{23}}\end{bmatrix}\begin{bmatrix}\frac{\partial P_{23}}{\partial x}\end{bmatrix} \quad (6.149)$$

$$= (1 + 0.125P_1)\begin{bmatrix}V_1V_2Y_{12}\sin(\delta_1 - \delta_2 - \phi_{1,2}) \\ V_1V_3Y_{13}\sin(\delta_1 - \delta_3 - \phi_{1,3})\end{bmatrix}$$
$$+2(P_{23} - 400)\begin{bmatrix}-V_2V_3Y_{23}\sin(\delta_2 - \delta_3 - \phi_{23}) \\ V_2V_3Y_{23}\sin(\delta_2 - \delta_3 - \phi_{23})\end{bmatrix} \quad (6.150)$$

Proceeding with the steepest gradient algorithm iterations yields the final constrained optimal generation scheduling:

$$\begin{bmatrix}P_1 \\ P_2 \\ P_3\end{bmatrix} = \begin{bmatrix}128.5 \\ 476.2 \\ 349.9\end{bmatrix} \text{ MW}$$

and P_{23} = 400 MW. The cost for this constrained scheduling is \$7,882/MW hr which is slightly greater than the non-constrained cost.

In the case where hard limits must be imposed, an alternate approach to enforcing the inequality constraints must be employed. In this approach, the inequality constraints are added as additional equality constraints with the inequality set equal to the limit (upper or lower) that is violated. This in essence introduces an additional set of Lagrangian multipliers. This is often referred to as the dual variable approach, because each inequality has the potential of resulting in two equalities: one for the upper limit and one for the lower limit. However, the upper and lower limit cannot be simultaneously violated; thus, out of the possible set of additional Lagrangian multipliers only one of the two will be included at any given operating point and thus the dual limits are mutually exclusive.

Example 6.16
Repeat Example 6.15 using the dual variable approach.

Solution 6.16 By introducing the additional equation

$$P_{23} = V_2V_3Y_{23}\cos(\delta_2 - \delta_3 - \phi_{23}) - V_2^2Y_{23}\cos\phi_{23} \quad = 0.400 \text{ per unit} \quad (6.151)$$

to the equality constraints adds an additional equation to the set of $g(x)$. Therefore an additional unknown must be added to the state vector x to yield a solvable set of equations (three equations in three unknowns). Either P_{G2} or P_{G3} can be chosen as the additional unknown. In this example, P_{G3} will be chosen. The new system Jacobian becomes:

$$\begin{bmatrix} \partial g \\ \partial x \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \frac{\partial g_1}{\partial x_3} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \frac{\partial g_2}{\partial x_3} \\ \frac{\partial g_3}{\partial x_1} & \frac{\partial g_3}{\partial x_2} & \frac{\partial g_3}{\partial x_3} \end{bmatrix} \tag{6.152}$$

where

$$\begin{aligned}
\frac{\partial g_1}{\partial x_1} &= V_2\left(V_1 Y_{12}\sin\left(\delta_2 - \delta_1 - \phi_{21}\right) + V_3 Y_{13}\sin\left(\delta_2 - \delta_3 - \phi_{23}\right)\right) \\
\frac{\partial g_1}{\partial x_2} &= -V_2 V_3 Y_{32}\sin\left(\delta_2 - \delta_3 - \phi_{23}\right) \\
\frac{\partial g_1}{\partial x_3} &= 0 \\
\frac{\partial g_2}{\partial x_1} &= -V_3 V_2 Y_{23}\sin\left(\delta_3 - \delta_2 - \phi_{32}\right) \\
\frac{\partial g_2}{\partial x_2} &= V_3 V_1 Y_{13}\sin\left(\delta_3 - \delta_1 - \phi_{31}\right) + V_2 Y_{23}\sin\left(\delta_3 - \delta_2 - \phi_{32}\right) \\
\frac{\partial g_2}{\partial x_3} &= 1 \\
\frac{\partial g_3}{\partial x_1} &= -V_2 V_3 Y_{23}\sin\left(\delta_2 - \delta_3 - \phi_{23}\right) \\
\frac{\partial g_3}{\partial x_2} &= V_2 V_3 Y_{23}\sin\left(\delta_2 - \delta_3 - \phi_{23}\right) \\
\frac{\partial g_3}{\partial x_3} &= 0
\end{aligned}$$

and

$$\begin{bmatrix} \partial g \\ \partial u \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}; \qquad \begin{bmatrix} \partial f \\ \partial u \end{bmatrix} = [1 + 0.025 P_{G2}]$$

Similar to Example 6.13, the chain rule is used to obtain $\begin{bmatrix} \partial f \\ \partial x \end{bmatrix}$:

$$\begin{bmatrix} \partial f \\ \partial x \end{bmatrix} = \begin{bmatrix} \partial C \\ \partial P_{G1} \end{bmatrix} \begin{bmatrix} \partial P_{G1} \\ \partial x \end{bmatrix} + \begin{bmatrix} \partial C \\ \partial P_{G3} \end{bmatrix} \begin{bmatrix} \partial P_{G3} \\ \partial x \end{bmatrix} \tag{6.153}$$

$$= (1 + 0.125 P_{G1}) \begin{bmatrix} V_1 V_2 Y_{12}\sin\left(\delta_1 - \delta_2 - \phi_{12}\right) \\ V_1 V_3 Y_{13}\sin\left(\delta_1 - \delta_3 - \phi_{13}\right) \\ 0 \end{bmatrix} + (1 + 0.050 P_{G3}) \times$$

$$\begin{bmatrix} V_3 V_2 Y_{32}\sin\left(\delta_3 - \delta_2 - \phi_{32}\right) \\ -V_3\left(V_1 Y_{13}\sin\left(\delta_3 - \delta_1 - \phi_{31}\right) + V_2 Y_{23}\sin\left(\delta_3 - \delta_2 - \phi_{32}\right)\right) \\ 0 \end{bmatrix} \tag{6.154}$$

Substituting these partial derivatives into the expression for C of equation (6.122) yields the same generation scheduling as Example 6.15.

6.4.2 State Estimation

In power system state estimation, the estimated variables are the voltage magnitudes and the voltage phase angles at the system buses. The input to the state estimator is the active and reactive powers of the system, measured either at the injection sites or on the transmission lines. The state estimator is designed to give the best estimates of the voltages and phase angles minimizing the effects of the measurement errors. Another consideration for the state estimator is to determine if a sufficient number of measurements are available to fully estimate the power system. This is the notion of system observability.

A set of specified measurements of a power system is said to be *observable* if the entire state vector of bus voltage magnitude and phase angles can be estimated from the set of available measurements. An unobservable system is one in which the set of measurements do not span the entire state space. The power system is observable if the matrix H_x in equation (6.47) has rank n (full rank), where the number of measurements m is greater than or equal to the number of system states n. A *redundant* measurement is one whose addition to the measurement does not increase the rank of the matrix H_x.

The observability of a power system can be determined by examining the measurement set and the structure of the power system. A *tree* is a set of measurements (either bus or line) that spans the entire set of power system buses. In other words, by graphically connecting the buses and lines that contribute to the set of measurements, the entire set of system buses can be connected by a single connected graph. A power system can be made observable by adding measurements at those lines that will connect disjoint trees.

Example 6.17

The SCADA system for the power network shown in Figure 6.7 reports the following measurements and variances:

7.1 The Power Method

The power method is one of the most commmon methods of finding the *dominant* eigenvalue of the $n \times n$ matrix A. The dominant eigenvalue is the largest eigenvalue in absolute value. Therefore if $\lambda_1, \lambda_2, \ldots, \lambda_n$ are eigenvalues of A, then λ_1 is the dominant eigenvalue of A if

$$|\lambda_1| > |\lambda_i| \tag{7.5}$$

for all $i = 2, \ldots, n$.

The power method is actually an approach to finding the eigenvector v_1 corresponding to the dominant eigenvalue of the matrix A. Once the eigenvector is obtained, then the eigenvalue can be extracted from the *Rayleigh quotient*:

$$\lambda = \frac{Av, v}{v, v} \tag{7.6}$$

The approach to finding the eigenvector v_1 is an interative approach. Therefore, from an initial guess vector v^0, a sequence of approximations v^k is constructed which hopefully converges as k goes to ∞. The iterative algorithm for the power method is straightforward:

The Power Method

1. Let $k = 0$ and choose v^0 to be a non-zero $n \times 1$ vector.
2. $w^{k+1} = Av^k$
3. $\alpha_{k+1} = \|w^{k+1}\|$
4. $v^{k+1} = \frac{w^{k+1}}{\alpha^{k+1}}$
5. if $\|v^{k+1} - v^k\| < \varepsilon$ then done. Else, $k = k + 1$, go to Step 2.

The division by the norm of the vector in Step 4 is not a necessary step, but it keeps the size of the values of the eigenvector close to 1. Recall that a scalar times an eigenvector of A is still an eigenvector of A, therefore scaling has no adverse consequence. However, without Step 4 and $\alpha^k = 1$ for all k, the values of the updated vector may increase or decrease to the extent that the computer accuracy is a ected.

Example 7.1

Use the power method to find the eigenvector corresponding to the dominant eigenvalue of the following matrix:

$$A = \begin{bmatrix} 6 & -2 \\ -8 & 3 \end{bmatrix}$$

Solution 7.1 Start with the initial guess $v^0 = [1 \;\; 1]^T$. Then

$$w^1 = A \quad v^0 = \begin{bmatrix} 4 \\ -5 \end{bmatrix}$$
$$\alpha^1 = \quad w^1 \quad = 6.4031$$
$$v^1 = \begin{bmatrix} 0.6247 \\ -0.7809 \end{bmatrix}$$

The second iteration follows:

$$w^2 = A \quad v^1 = \begin{bmatrix} 5.3099 \\ -7.3402 \end{bmatrix}$$
$$\alpha^2 = \quad w^2 \quad = 9.0594$$
$$v^2 = \begin{bmatrix} 0.5861 \\ -0.8102 \end{bmatrix}$$

Continuing to convergence yields the eigenvalue:

$$v^* = \begin{bmatrix} 0.5851 \\ -0.8110 \end{bmatrix}$$

From the eigenvector, the corresponding eigenvalue is calculated

$$\lambda = \frac{\begin{bmatrix} 0.5851 & -0.8110 \end{bmatrix} \begin{bmatrix} 6 & -2 \\ -8 & 3 \end{bmatrix} \begin{bmatrix} 0.5851 \\ -0.8110 \end{bmatrix}}{\begin{bmatrix} 0.5851 & -0.8110 \end{bmatrix} \begin{bmatrix} 0.5851 \\ -0.8110 \end{bmatrix}} = \frac{8.7720}{1} = 8.7720$$

which is exactly the largest eigenvalue of A (the smaller eigenvalue is 0.2280).

To see why the power method converges to the dominant eigenvector, let the initial guess vector v^0 be expressed as the linear combination:

$$v^0 = \sum_{i=1}^{n} \beta_i v_i \tag{7.7}$$

where v_i are the actual eigenvectors of A and the coe cients β_i are chosen to make equation (7.7) hold. Then applying the power method (without loss of generality it can be assumed that $\alpha^k = 1$ for all k) yields

$$v^{k+1} = Av^k = A^2 v^{k-1} = \ldots = A^{k+1} v^0$$
$$= \sum_{i=1}^{n} \lambda_i^{k+1} \beta_i v_i = \lambda_1^{k+1} \left(\beta_1 v_1 + \sum_{i=2}^{n} \left(\frac{\lambda_i}{\lambda_1} \right)^{k+1} \beta_i v_i \right)$$

Because

$$\left|\frac{\lambda_i}{\lambda_1}\right| < 1 \quad i = 2, \ldots, n$$

then as k successively increases, these terms go to zero, and only the component corresponding to v_1 is left.

The power method will fail if there are two largest eigenvalues of the same absolute magnitude. Recall that the eigenvalues of a real matrix A are in general complex and occur in conjugate pairs (which necessarily have the same absolute value). Therefore if the largest eigenvalue of A is not real, then the power method will certainly fail to converge. For this reason, it is sensible to apply the power method only to matrices whose eigenvalues are known to be real. One class of real eigenvalue matrices are symmetric matrices.

There are also cases in which the power method may not converge to the eigenvector corresponding to the dominant eigenvalue. This would occur in the case in which $\beta_1 = 0$. This implies that the initial guess v^0 contains no component of the eigenvector v_1. In this case, the method will converge to the eigenvector contained in the decomposition of v^0 of the next largest eigenvalue.

The rate of convergence of the power method is determined by the ratio $|\frac{\lambda_2}{\lambda_1}|$. Thus if $|\lambda_2|$ is only slightly smaller than $|\lambda_1|$, then the power method will converge slowly and a large number of iterations will be required to meet the required accuracy.

There are several extensions to the power method. For example, if instead of the dominant eigenvalue, the smallest eigenvalue was desired, then the power method can be applied to A^{-1}. Since the eigenvalues of A^{-1} are $\frac{1}{\lambda_n}, \ldots, \frac{1}{\lambda_1}$, the inverse power method should converge to $\frac{1}{\lambda_n}$.

Another extension is the spectral shift. This approach uses the fact that the eigenvalues of $A - aI$ are $\lambda_1 - a, \ldots, \lambda_n - a$. Thus having computed the first eigenvalue λ_1, the power method can be reapplied using the shifted matrix $A - \lambda_1 I$. This reduces the first eigenvalue to zero and the power method now converges to the largest in absolute value of $\lambda_2 - \lambda_1, \ldots, \lambda_n - \lambda_1$.

7.2 The QR Algorithm

Many methods for solving the eigenvalue problem are based on a sequence of similarity transformations with orthogonal matrices. Thus, if P is any nonsingular matrix, then the matrices A and PAP^{-1} have the same eigenvalues. Furthermore, if v is an eigenvector of A, then Pv is an eigenvector of PAP^{-1}. If the matrix P is othogonal, then the condition of the eigenproblem is not a ected. This is the basis for the similarity transformation methods.

The QR method [20], [52], [53] is one of the most widely used decomposition methods for calculating eigenvalues of matrices. It uses a sequence of orthogonal similarity transformations [13] [28] such that $A = A_0, A_1, A_2, \ldots$ is computed by

$$A_i = Q_i R_i, \quad R_i Q_i = A_{i+1}, \quad i = 0, 1, 2, \ldots,$$

Similar to the LU factorization, the matrix A can also be factored into two matrices such that

$$A = QR \tag{7.8}$$

where Q is a unitary matrix and R is an upper triangular matrix. The matrix Q is *unitary* if

$$QQ^* = Q^*Q = I \tag{7.9}$$

where ( ) denotes complex conjugate transpose.

Examples of unitary matrices are

$$Q_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Q_2 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

It also follows that the inverse of a unitary matrix is also its conjugate transpose, i.e.,

$$Q^{-1} = Q^*$$

This decomposition yields the column vectors $[a_1, a_2, \ldots, a_n]$ of A and column vectors $[q_1, q_2, \ldots, q_n]$ of Q such that

$$a_k = \sum_{i=1}^{k} r_{ik} q_i, \quad k = 1, \ldots, n \tag{7.10}$$

The column vectors $a_1, a_2, \ldots, a_n$ must be orthonormalized from the left to right into an orthonormal basis $q_1, q_2, \ldots, q_n$.

In the implementation of the QR algorithm, it is common practice to transform A into a Hessenberg matrix H having the same eigenvalues and then apply the QR matrix to H. In the end, the matrix becomes upper triangular and the eigenvalues can be read o of the diagonal. A Hessenberg matrix is essentially an upper triangular matrix with one extra set of non-zero elements directly below the diagonal. The reason for reducing A to a Hessenberg matrix is that this greatly reduces the total number of operations required for the QR algorithm.

The Householder method is one method used to reduce A to a Hessenberg matrix. For each $n \times n$ matrix A, there exist $n - 2$ Householder matrices $H_1, H_2, \ldots, H_{n-2}$, such that for

$$Q = H_{n-2} \ldots H_2 H_1$$

the matrix

$$P = Q^* A Q$$

is a Hessenberg matrix [27]. A matrix H is a Householder matrix if

$$H = I - 2\frac{vv^*}{v^*v}$$

Note that Householder matrices are also unitary matrices. The vector v is chosen to satisfy

$$v_i = a_i \pm e_i \quad a_i \quad _2 \tag{7.11}$$

where the choice of sign is based upon the requirement that $v \ _2$ should not be too small, e_i is the i^{th} column of I, and a_i is the i^{th} column of A.

Example 7.2

Find the QR decomposition of the matrix A:

$$A = \begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix}$$

Solution 7.2 The first transformation will be applied to zero out the first column of A below the subdiagonal, thus

$$\begin{aligned} v_1 &= a_1 + e_1 \quad a_1 \quad _2 \\ &= \begin{bmatrix} 1 \\ 2 \\ 4 \\ 9 \end{bmatrix} + 10.0995 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 11.0995 \\ 2.0000 \\ 4.0000 \\ 9.0000 \end{bmatrix} \end{aligned}$$

leading to

$$\begin{aligned} H_1 &= I - 2\frac{v_1 v_1^*}{(v_1^* v_1)} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \frac{2}{224.1990} \begin{bmatrix} 11.0995 \\ 2.0000 \\ 4.0000 \\ 9.0000 \end{bmatrix} \begin{bmatrix} 11.0995 & 2.0000 & 4.0000 & 9.0000 \end{bmatrix} \\ &= \begin{bmatrix} -0.0990 & -0.1980 & -0.3961 & -0.8911 \\ -0.1980 & 0.9643 & -0.0714 & -0.1606 \\ -0.3961 & -0.0714 & 0.8573 & -0.3211 \\ -0.8911 & -0.1606 & -0.3211 & 0.2774 \end{bmatrix} \end{aligned}$$

and

$$H_1 A = \begin{bmatrix} -10.0995 & -3.4655 & -9.0103 & -8.1192 \\ 0 & -0.1650 & -0.3443 & 0.0955 \\ 0 & 0.6700 & 0.3114 & 2.1910 \\ 0 & -3.2425 & -3.5494 & -9.0702 \end{bmatrix}$$

The second iteration will operate on the part of the transformed matrix that excludes the first column and row. Therefore

$$\begin{aligned} v_2 &= a_2 + e_2 \; a_{2\;2} \\ &= \begin{bmatrix} -0.1650 \\ 0.6700 \\ -3.2425 \end{bmatrix} + 3.3151 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 3.1501 \\ 0.6700 \\ -3.2425 \end{bmatrix} \end{aligned}$$

which results in

$$\begin{aligned} H_2 &= I - 2 \frac{v_2 v_2^*}{(v_2^* v_2)} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.0498 & -0.2021 & 0.9781 \\ 0 & -0.2021 & 0.9570 & 0.2080 \\ 0 & 0.9781 & 0.2080 & -0.0068 \end{bmatrix} \end{aligned}$$

and

$$H_2 H_1 A = \begin{bmatrix} -10.0995 & -3.4655 & -9.0103 & -8.1192 \\ 0 & -3.3151 & -3.5517 & -9.3096 \\ 0 & 0 & -0.3708 & 0.1907 \\ 0 & 0 & -0.2479 & 0.6108 \end{bmatrix}$$

Continuing the process yields

$$v_3 = \begin{bmatrix} 0.0752 \\ -0.2479 \end{bmatrix}$$

$$H_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.8313 & 0.5558 \\ 0 & 0 & 0.5558 & -0.8313 \end{bmatrix}$$

which results in

$$R = H_3 H_2 H_1 A = \begin{bmatrix} -10.0995 & -3.4655 & -9.0103 & -8.1192 \\ 0 & -3.3151 & -3.5517 & -9.3096 \\ 0 & 0 & -0.4460 & 0.4980 \\ 0 & 0 & 0 & -0.4018 \end{bmatrix}$$

and

$$Q = H_1 H_2 H_3 = \begin{bmatrix} -0.0990 & -0.8014 & -0.5860 & -0.0670 \\ -0.1980 & -0.0946 & 0.2700 & -0.9375 \\ -0.3961 & -0.4909 & 0.7000 & 0.3348 \\ -0.8911 & 0.3283 & -0.3060 & 0.0670 \end{bmatrix}$$

It can be verified that $A = QR$ and further that $Q^* = Q^{-1}$.

The elimination by QR decomposition can be considered as an alternative to Gaussian elimination. However, the number of multiplications and divisions required is more than twice the number required for Gaussian elimination. Therefore QR decomposition is seldom used for the solution of linear systems, but it does play an important role in the calculation of eigenvalues.

Although the eigenvalue problem gives rise to a simple set of algebraic equations to determine the solution to

$$det\,(A - \lambda I) = 0$$

the practical problem of solving this equation is di cult. Computing the roots of the characteristic equation or the nullspace of a matrix is a process that is not well suited for computers. In fact, no generalized direct process exists for solving the eigenvalue problem in a finite number of steps. Therefore iterative methods for calculation must be relied upon to produce a series of successively improved approximations to the eigenvalues of a matrix.

The QR method is commonly used to calculate the eigenvalues and eigenvectors of full matrices. As developed by Francis [13], the QR method produces a series of similarity transformations

$$A_k = Q_k^* A_{k-1} Q_k \quad Q_k^* Q_k = I \tag{7.12}$$

where the matrix A_k is similar to A. The QR decomposition is repeatedly performed and applied to A as the subdiagonal elements are iteratively driven to zero. At convergence, the eigenvalues of A in descending order by magnitude appear on the diagonal of A_k.

Example 7.3

Find the eigenvalues and eigenvectors of the matrix of Example 7.2.

Solution 7.3 The first objective is to find the eigenvalues of the matrix A using the QR method. From Example 7.2, the first QR factorization yields the Q_0 matrix

$$Q_0 = \begin{bmatrix} -0.0990 & 0.8014 & 0.5860 & -0.0670 \\ -0.1980 & 0.0946 & -0.2700 & -0.9375 \\ -0.3961 & 0.4909 & -0.7000 & 0.3348 \\ -0.8911 & -0.3283 & 0.3060 & 0.0670 \end{bmatrix}$$

Using the given A matrix as A_0, the first update A_1 is found by

$$A_1 = Q_0^* A_0 Q_0$$
$$= \begin{bmatrix} 12.4902 & -10.1801 & -1.1599 & 0.3647 \\ -10.3593 & -0.9987 & -0.5326 & -1.2954 \\ 0.2672 & 0.3824 & -0.4646 & 0.1160 \\ 0.3580 & 0.1319 & -0.1230 & -0.0269 \end{bmatrix}$$

The QR factorization of A_1 yields

$$Q_1 = \begin{bmatrix} -0.7694 & -0.6379 & -0.0324 & -0.0006 \\ 0.6382 & -0.7660 & -0.0733 & 0.0252 \\ -0.0165 & 0.0687 & -0.9570 & -0.2812 \\ -0.0221 & 0.0398 & -0.2786 & 0.9593 \end{bmatrix}$$

and the A_2 matrix becomes

$$A_2 = Q_1^* A_1 Q_1$$
$$= \begin{bmatrix} 17.0913 & 4.8455 & -0.2315 & -1.0310 \\ 4.6173 & -5.4778 & -1.8116 & 0.6064 \\ -0.0087 & 0.0373 & -0.5260 & -0.1757 \\ 0.0020 & -0.0036 & 0.0254 & -0.0875 \end{bmatrix}$$

Note that the elements below the diagonals are slowly decreasing to zero. This process is carried out until the final A matrix is obtained:

$$A_* = \begin{bmatrix} 18.0425 & 0.2133 & -0.5180 & -0.9293 \\ 0 & -6.4172 & -1.8164 & 0.6903 \\ 0 & 0 & -0.5269 & -0.1972 \\ 0 & 0 & 0 & -0.0983 \end{bmatrix} \tag{7.13}$$

The eigenvalues are on the diagonals of A_* and are in decreasing order by magnitude. Thus the eigenvalues are

$$\lambda_{1,\ldots,4} = \begin{bmatrix} 18.0425 \\ -6.4172 \\ -0.5269 \\ -0.0983 \end{bmatrix}$$

The next step is to find the eigenvectors associated with each eigenvalue. Recall that

$$Av_i = \lambda_i v_i \tag{7.14}$$

for each eigenvalue and corresponding eigenvector $i = 1, \ldots, n$. Equation (7.14) may also be written as

$$Av_i - \lambda_i v_i = 0$$

In other words, the matrix defined by $A - \lambda_i I$ is singular; thus, only three of its rows (or columns) are independent. This fact can be used to determine the eigenvectors once the eigenvalues are known. Since $A - \lambda_i I$ is not of full rank, one of the elements of the eigenvector v_i can be chosen arbitrarily. To start, partition $A - \lambda_i I$ as

$$A - \lambda_i I = \begin{bmatrix} a_{11} & a_{1,2n} \\ a_{2n,1} & a_{2n,2n} \end{bmatrix}$$

where a_{11} is a scalar, $a_{1,2n}$ is a $1 \times (n-1)$ vector, $a_{2n,1}$ is a $(n-1) \times 1$ vector, and $a_{2n,2n}$ is an $(n-1) \times (n-1)$ matrix of rank $(n-1)$. Then let $v_i(1) = 1$ and solve for the remaining portion of the eigenvector as

$$\begin{bmatrix} v_i(2) \\ v_i(3) \\ \vdots \\ v_i(n) \end{bmatrix} = -a_{2n,2n}^{-1} a_{2n,1} v_i(1) \tag{7.15}$$

Now update $v_i(1)$ from

$$v_i(1) = -\frac{1}{a_{11}} a_{2n,1} \begin{bmatrix} v_i(2) \\ v_i(3) \\ \vdots \\ v_i(n) \end{bmatrix}$$

Then the eigenvector corresponding to λ_i is

$$v_i = \begin{bmatrix} v_i(1) \\ v_i(2) \\ \vdots \\ v_i(n) \end{bmatrix}$$

The last step is to normalize the eigenvector; therefore,

$$v_i = \frac{v_i}{v_i}$$

Thus, for the vector of eigenvalues

$$\Lambda = \begin{bmatrix} 18.0425 & -6.4172 & -0.5269 & -0.0983 \end{bmatrix}$$

the corresponding eigenvectors are:

$$\begin{bmatrix} 0.4698 \\ 0.2329 \\ 0.5800 \\ 0.6234 \end{bmatrix}, \begin{bmatrix} 0.6158 \\ 0.0539 \\ 0.2837 \\ -0.7330 \end{bmatrix}, \begin{bmatrix} 0.3673 \\ -0.5644 \\ -0.5949 \\ 0.4390 \end{bmatrix}, \begin{bmatrix} 0.0932 \\ 0.9344 \\ -0.2463 \\ -0.2400 \end{bmatrix}$$

7.2.1 Shifted QR

The QR iterations can converge very slowly in many instances. However, if some information about one or more of the eigenvalues is known a priori, then a variety of techniques can be applied to speed up convergence of the iterations. One such technique is the *shifted* QR method, in which a shift σ is introduced at each iteration such that the QR factorization at the k^{th} is performed on

$$A_k - \sigma I = Q_k R_k$$

and

$$A_{k+1} = Q_k^* (A_k - \sigma I) Q_k + \sigma I$$

If σ is a good estimate of an eigenvalue, then the $(n, n-1)$ entry of A_k will converge rapidly to zero, and the (n, n) entry of A_k will converge to the eigenvalue closest to σ_k. Once this has occurred, an alternate shift can be applied.

Example 7.4
Repeat Example 7.3 using shifts.

Solution 7.4 Start with using a shift of $\sigma = 15$. This is near the 18.0425 eigenvalue; so, convergence to that particular eigenvalue should be rapid. Starting with the original A matrix as A_0, the QR factorization of $A_0 - \sigma I$ yields

$$Q_0 = \begin{bmatrix} -0.8124 & 0.0764 & 0.2230 & 0.5334 \\ 0.1161 & -0.9417 & -0.0098 & 0.3158 \\ 0.2321 & 0.2427 & -0.7122 & 0.6164 \\ 0.5222 & 0.2203 & 0.6655 & 0.4856 \end{bmatrix}$$

and the update $A_1 = Q_0^* (A_0 - \sigma I) Q_0 + \sigma I$

$$A_1 = \begin{bmatrix} -4.9024 & 0.8831 & -1.6174 & 2.5476 \\ -0.2869 & 0.0780 & -0.1823 & 1.7775 \\ -2.9457 & 0.5894 & -1.5086 & 2.3300 \\ 2.5090 & 1.0584 & 3.1975 & 17.3330 \end{bmatrix}$$

The eigenvalue of interest ($\lambda = 18.00425$) will now appear in the lower right corner since as the iterations progress $A_{k+1}(n, n) - \sigma$ will be the smallest diagonal in magnitude. Recall that the eigenvalues are ordered on the diagonal from largest to smallest and since the largest eigenvalue is "shifted" by σ, it will now have the smallest magnitude. The convergence can be further increased by updating σ at each iteration, such that $\sigma_{k+1} = A_{k+1}(n, n)$. The iterations proceed as in Example 7.3.

7.2.2 Deflation

The speed of convergence of the QR method for calculating eigenvalues depends greatly on the location of the eigenvalues with respect to one another. The matrix $A - \sigma I$ has the eigenvalues $\lambda_i - \sigma$ for $i = 1, \ldots, n$. If σ is chosen as an approximate value of the smallest eigenvalue λ_n, then $\lambda_n - \sigma$ becomes small. This will speed up the convergence in the last row of the matrix, since

$$\frac{|\lambda_n - \sigma|}{|\lambda_{n-1} - \sigma|} \quad 1$$

Once the elements of the last row are reduced to zero, the last row and column of the matrix may be neglected. This implies that the smallest eigenvalue is "deflated" by removing the last row and column. The procedure can then be repeated on the remaining $(n-1) \times (n-1)$ matrix with the shift σ chosen close to λ_{n-1}. Using the shift and deflation in combination can significantly improve convergence. Additionally, if only one eigenvalue is desired of a particular magnitude, this eigenvalue can be isolated via the shift method. After the last row has been driven to zero, the eigenvalue can be obtained and the remainder of the QR iterations abandoned.

7.3 Arnoldi Methods

In large interconnected systems, it is either impractical or intractable to find all of the eigenvalues of the system state matrix due to restrictions on computer memory and computational speed. The Arnoldi method has been developed as an algorithm that iteratively computes k eigenvalues of an $n \times n$ matrix A, where k is typically much smaller than n. This method therefore bypasses many of the constraints imposed by large matrix manipulation required by methods such as the QR decomposition. If the k eigenvalues are chosen selectively, they can yield rich information about the system under consideration, even without the full set of eigenvalues. The Arnoldi method was first developed in [2], but su ered from poor numerical properties such as loss of orthogonality and slow convergence. Several modifications to the Arnoldi method have overcome these shortcomings. The Modified Arnoldi Method (MAM) has been used frequently in solving eigenvalue problems in power system applications [29], [51]. This approach introduced preconditioning and explicit restart techniques to retain orthogonality. Unfortunately however, an explicit restart will often discard useful information. The restart problem was solved by using implicitly shifted QR steps [45] in the Implicitly Restarted Arnoldi (IRA) method. Several commercial software packages have been developed around the IRA method, including the well known ARPACK and the Matlab `speig` routines.

The basic approach of the Arnoldi method is to iteratively update a low order matrix H whose eigenvalues successively approximate the selected eigenvalues of the larger A matrix, such that

$$AV = VH; \quad V^*V = I \tag{7.16}$$

where V is an $n \times k$ matrix and H is a $k \times k$ Hessenberg matrix. As the method progresses, the eigenvalues of A are approximated by the diagonal entries of H yielding

$$HV_i = V_i D \tag{7.17}$$

where V_i is a $k \times k$ matrix whose columns are the eigenvalues of H (approximating the eigenvectors of A) and D is a $k \times k$ matrix whose diagonal entries are the eigenvalues of H (approximating the eigenvalues of A). The Arnoldi method is an orthogonal projection method onto a *Krylov* subspace.

The Arnoldi procedure is an algorithm for building an orthogonal basis of the Krylov subspace. One approach is given as:

The k-step Arnoldi Factorization

Starting with a vector v_1 of unity norm, for $j = 1, \ldots, k$ compute:

1. $H(i,j) = v_i^T A v_j$ for $i = 1, \ldots, j$
2. $w_j = Av_j - \sum_{i=1}^{j} H(i,j) v_i$
3. $H(j+1, j) = \|w_j\|_2$
4. If $H(j+1, j) = 0$, then stop
5. $v_{j+1} = \frac{w_j}{H(j+1,j)}$

At each step, the algorithm multiplies the previous Arnoldi vector v_j by A and then orthonormalizes the resulting vector w_j against all previous v_i's. The k-step Arnoldi factorization is shown in Figure 7.1, and is given by

$$AV_k = V_k H_k + w_k e_k^T \tag{7.18}$$

The columns $V = [v_1, v_2, \ldots, v_k]$ form an orthonormal basis for the Krylov subspace and H is the orthogonal projection of A onto this space. It is desirable for $\|w_k\|$ to become small because this indicates that the eigenvalues of H are accurate approximations to the eigenvalues of A. However, this "convergence" often comes at the price of numerical orthogonality in V. Therefore, the k-step Arnoldi factorization is "restarted" to preserve orthogonality.

Implicit restarting provides a means to extract rich information from very large Krylov subspaces while avoiding the storage and poor numerical properties associated with the standard approach. This is accomplished by continually compressing the information into a fixed size k-dimensional subspace, by using a shifted QR mechanism. A $(k + p)$-step Arnoldi factorization

$$AV_{k+p} = V_{k+p} H_{k+p} + w_{k+p} e_{k+p}^T \tag{7.19}$$

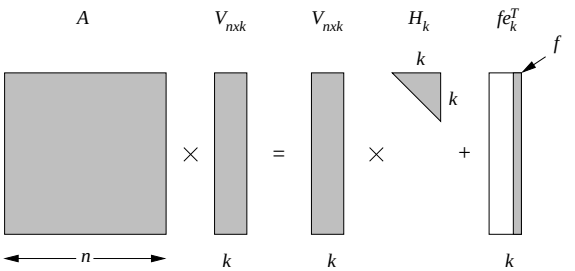


FIGURE 7.1
A k-step Arnoldi factorization

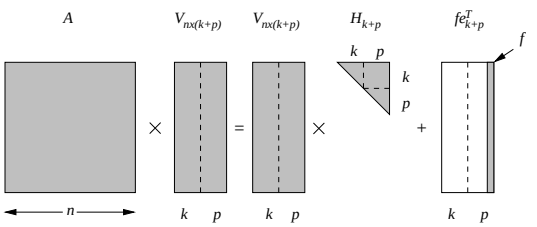


FIGURE 7.2
A $(k + p)$-step Arnoldi factorization

is compressed to a factorization of length k that retains the eigen-information of interest. This is accomplished using QR steps to apply p shifts to yield

$$A\hat{V}_{k+p} = \hat{V}_{k+p}\hat{H}_{k+p} + \hat{w}_{k+p} \tag{7.20}$$

where $\hat{V}_{k+p} = V_{k+p}Q$, $\hat{H}_{k+p} = Q^* H_{k+p}Q$, and $\hat{w}_{k+p} = w_{k+p}e_{k+p}^T Q$. It may be shown that the first $k - 1$ entries of the vector $e_{k+p}^T Q$ are zero [46]. Equating the first k columns on both sides yields an updated k-step Arnoldi factorization. This now provides the "restart" vectors for extending the k-step Arnoldi factorization to the $k + p$-step Arnoldi factorization, shown in Figure 7.2.

The implicitly restarted Arnoldi algorithm consists of three main steps: initialization, iteration/refinement, and final calculation of the eigenvalues and eigenvectors.

Implicity Restarted Arnoldi Algorithm

1. Initialization

Using the vector v_1 as a starting vector, generate a k-step Arnoldi factorization. At each step k of the factorization, the vector V_k is augmented by a vector v_k satisfying equation (7.18). Note that H_k is a Hessenberg matrix. The shaded regions in Figure 7.1 represent non-zero entries. The unshaded region of fe_k^T is a zero matrix of $(k - 1)$ columns. The last column of fe_k^T is f. The Arnoldi factorization is entirely dependent on the choice of initial vector v_1.

2. Iteration/Refinement

   (a) Extend the k-step factorization by p steps.

   Each of the p additions represents an eigenvalue/eigenvector that can be discarded at the end of the iteration if it does not meet the chosen criteria. In general, the choice of p is a trade-o between the length of factorization that may be tolerated and the rate of convergence. For most problems, the size of p is determined experimentally. The only requirement is that $1 \leq p \leq n - k$.

   (b) Calculate eigenvalues of $H\ Tc\phi \quad Tm\phi\ .\quad Tm\phi\ \phi\phi\ T \quad Tf\phi\ .\quad .$

of minimizing the error between the actual time-varying function and the proposed function by estimating the magnitude, phase, and damping parameters of the fitting function. In the problem of estimating a nonlinear waveform by a series of functions, the minimization function is given by:

$$\min f = \sum_{i=1}^{N} \left[\sum_{k=1}^{n} \left[a_k e^{(b_k t_i)} \cos(\omega_k t_i + \theta_k) \right] - y_i \right]^2 \tag{7.33}$$

where n is the number of desired modes of the approximating waveform, N is the number of data samples, y_i is the sampled waveform, and

$$[a_1\ b_1\ \omega_1\ \theta_1,\ \ldots, a_n\ b_n\ \omega_n\ \theta_n]^T$$

are the parameters to be estimated.

There are several approaches to estimating the modal content of a time varying waveform. The Prony method is well-known and widely used in power systems applications. The matrix pencil approach was introduced for extracting poles from antennas' electromagnetic transient responses. The Levenberg-Marquardt iteratively updates the modal parameters by an analytic optimization to minimize the error between the resulting waveform and the input data.

7.5.1 Prony Method

One approach to estimate the various parameters is the *Prony method* [23]. This method is designed to directly estimate the parameters for the exponential terms by fitting the function

$$\hat{y}(t) = \sum_{i=1}^{n} A_i e^{\sigma_i t} \cos(\omega_i t + \phi_i) \tag{7.34}$$

to an observed measurement for $y(t)$, where $y(t)$ consists of N samples

$$y(t_k) = y(k), \quad k = 0, 1, \ldots, N-1$$

that are evenly spaced by a time interval Δt. Since the measurement signal $y(t)$ may contain noise or dc o set, it may have to be conditioned before the fitting process in applied.

The basic Prony method is summarized as

Prony Method

1. Construct a discrete linear prediction model from the measurement set
2. Find the roots of the characteristic polynomial of the model
3. Using the roots as the complex modal frequencies for the signal, determine the amplitude and phase for each mode

These steps are performed in the z-domain, translating the eigenvalues to the s-domain as a final step.

Note that equation (7.34) can be recast in complex exponential form as:

$$\hat{y}(t) = \sum_{i=1}^{n} B_i e^{\lambda_i t} \tag{7.35}$$

which can be translated to

$$\hat{y}(k) = \sum_{i=1}^{n} B_i z_i^k \tag{7.36}$$

where

$$z_i = e^{(\lambda_i \Delta t)} \tag{7.37}$$

The system eigenvalues λ can be found from the discrete modes by

$$\lambda_i = \frac{ln\,(z_i)}{\Delta t} \tag{7.38}$$

The z_i are the roots of the n-th order polynomial

$$z^n - \left(a_1 z^{n-1} + a_2 z^{n-2} + \ldots + a_n z^0\right) = 0 \tag{7.39}$$

where the a_i coe cients are unknown and must be calculated from the measurement vector as:

$$\begin{bmatrix} y(n-1) & y(n-2) & \ldots & y(0) \\ y(n-0) & y(n-1) & \ldots & y(1) \\ \vdots & \vdots & \vdots & \vdots \\ y(N-2) & y(N-3) & \ldots & y(N-n-1) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y(n) \\ y(n+1) \\ \ldots \\ y(N-1) \end{bmatrix} \tag{7.40}$$

Note that this is a system of N equations in n unknowns and therefore must be solved by the least squares method to find the best fit.

Once z_i has been computed from the roots of equation (7.39), then the eigenvalues λ_i can be calculated from equation (7.38). The next step is to find the B_i that produces $\hat{y}(k) = y(k)$ for all k. This leads to the following relationship:

$$\begin{bmatrix} z_1^0 & z_2^0 & \ldots & z_n^0 \\ z_1^1 & z_2^1 & \ldots & z_n^1 \\ \vdots & \vdots & \vdots & \vdots \\ z_1^{N-1} & z_2^{N-1} & \ldots & z_n^{N-1} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} y(0) \\ y(1) \\ \vdots \\ y(N-1) \end{bmatrix} \tag{7.41}$$

which can be succinctly expressed as

$$ZB = Y \tag{7.42}$$

Note that the matrix B is $n \times N$; therefore, equation (7.42) must also be solved by the least squares method. The estimating waveform $\hat{y}(t)$ is then calculated from equation (7.35). The reconstructed signal $\hat{y}(t)$ will usually not fit $y(t)$ exactly. An appropriate measure for the quality of this fit is a "signal to noise ratio (SNR)" given by

$$\text{SNR} = 20 \log \frac{\hat{y} - y}{y} \tag{7.43}$$

where the SNR is given in decibels (dB).

Since the fit for this method may be inexact, it is desirable to have control over the level of error between the fitting function and the original waveform. In this case, a nonlinear least squares can provide improved results.

7.5.2 The Matrix Pencil Method

The Prony method described in the previous section is a "polynomial" method in that it includes the process of finding the poles z_i of a characteristic polynomial. The matrix pencil (MP) method produces a matrix whose roots provide z_i. The poles are found as the solution of a generalized eigenvalue problem [24]-[41]. The matrix pencil is given by

$$[Y_2] - \lambda[Y_1] = [Z_1][B]\{[Z_0] - \lambda[I]\}[Z_2] \tag{7.44}$$

where

$$[Y] = \begin{bmatrix} y(0) & y(1) & \dots & y(L) \\ y(1) & y(2) & \dots & y(L+1) \\ \vdots & \vdots & & \vdots \\ y(N-L) & y(N-L+1) & \dots & y(N) \end{bmatrix} \tag{7.45}$$

$$[Z_0] = \text{diag}[z_1, z_2, \dots, z_n] \tag{7.46}$$

$$[Z_1] = \begin{bmatrix} 1 & 1 & \dots & 1 \\ z_1 & z_2 & \dots & z_n \\ \vdots & \vdots & & \vdots \\ z_1^{(N-L-1)} & z_2^{(N-L-2)} & \dots & Z_n^{(N-L-1)} \end{bmatrix} \tag{7.47}$$

$$[Z_2] = \begin{bmatrix} 1 & z_1 & \dots & z_1^{L-1} \\ 1 & z_2 & \dots & z_2^{L-1} \\ \vdots & \vdots & & \vdots \\ 1 & Z_n & \dots & Z_n^{l-1} \end{bmatrix} \tag{7.48}$$

$[B]$ = matrix of residuals

$[I]$ = $n \times n$ identity matrix

n = desired number of eigenvalues

L = pencil parameter, such that $n \le L \le N - n$

Matrix Pencil Method

1. Choose L such that $n \leq L \leq N - n$
2. Construct the matrix $[Y]$
3. Perform a singular value decomposition of $[Y]$ to obtain

$$[Y] = [U][S][V]^T \tag{7.49}$$

where $[U]$ and $[V]$ are unitary matrices and contain the eigenvectors of $[Y][Y]^T$ and $[Y]^T[Y]$ respectively.

4. Construct the matrices $[V_1]$ and $[V_2]$ such that

$$V_1 = [v_1 \;\; v_2 \;\; v_3 \;\; \ldots \;\; v_{n-1}] \tag{7.50}$$
$$V_2 = [v_2 \;\; v_3 \;\; v_4 \;\; \ldots \;\; v_n] \tag{7.51}$$

where v_i is the i-th right singular vector of V.

5. Construct $[Y_1]$ and $[Y_2]$

$$[Y_1] = [V_1]^T[V_1]$$
$$[Y_2] = [V_2]^T[V_1]$$

6. The desired poles z_i may be found as the generalized eigenvalues of the matrix pair $\{[Y_2]; [Y_1]\}$.

From this point, the remainder of the algorithm follows that of the Prony method to calculate the eigenvalues λ and the residual matrix B.

If the pencil parameter L is chosen such that $L = N/2$, then the performance of the method is very close to the optimal bound [24].

It has been shown that under noise, the statistical variance of the poles found from the Matrix Pencil method is always less than that of the Prony Method [24].

7.5.3 The Levenberg-Marquardt Method

The nonlinear least squares for data fitting applications has the general form

$$\text{minimize } f(x) = \sum_{k=1}^{N} [\hat{y}(x, t_i) - y_i]^2 \tag{7.52}$$

where y_i is the output of the system at time t_i, and x is the vector of magnitudes, phases, and damping coe cients of equation (7.34), which arise from the eigenvalues of the state matrix of the system.

To find the minimum of $f(x)$, the same procedure for developing the Newton-Raphson iteration is applied. The function $f(x)$ is expanded about some x_0 by the Taylor series:

$$f(x) \approx f(x_0) + (x - x_0)^T f'(x_0) + \frac{1}{2}(x - x_0)^T f''(x_0)(x - x_0) + \ldots \quad (7.53)$$

where

$$f'(x) = \frac{\partial f}{\partial x_j} \quad \text{for } j = 1, \ldots, n$$

$$f''(x) = \frac{\partial^2 f}{\partial x_j \partial x_k} \quad \text{for } j, k = 1, \ldots, n$$

If the higher order terms in the Taylor's expansion are neglected, then minimizing the quadratic function on the right hand side of equation (7.53) yields

$$x_1 = x_0 - [f''(x_0)]^{-1} f'(x_0) \quad (7.54)$$

which yields an approximation for the minimum of the function $f(x)$. This is also one Newton-Raphson iteration update for solving the necessary minimization condition

$$f'(x) = 0$$

The Newton-Raphson equation (7.54) may be rewritten as the iterative linear system

$$A(x_k)(x_{k+1} - x_k) = g(x_k) \quad (7.55)$$

where

$$g_j(x) = -\frac{\partial f}{\partial x_j}(x)$$

$$a_{jk}(x) = \frac{\partial^2 f}{\partial x_j \partial x_k}(x)$$

and the matrix A is the system Jacobian (or similar iterative matrix).

The derivatives of equation (7.52) are

$$\frac{\partial f}{\partial x_j}(x) = 2\sum_{k=1}^{N} [\hat{y}_k - y_k] \frac{\partial \hat{y}_i}{\partial x_j}(x)$$

and

$$\frac{\partial^2 f}{\partial x_j \partial x_k}(x) = 2\sum_{k=1}^{N} \left\{ \frac{\partial \hat{y}_k}{\partial x_j}(x) \frac{\partial \hat{y}_i}{\partial x_k}(x) + [\hat{y}_k - y_k] \frac{\partial^2 \hat{y}_k}{\partial x_j \partial x_k}(x) \right\}$$

In this case, the matrix element a_{jk} contains second derivatives of the functions $\hat{y}_i$. These derivatives are multiplied by the factor $[\hat{y}_i(x) - y_i]$ and will become small during the minimization of f. Therefore the argument can be

made that these terms can be neglected during the minimization process. Note that if the method converges, it will converge regardless of whether the exact Jacobian is used in the iteration. Therefore the iterative matrix A can be simplified as

$$a_{jk} = 2\sum_{i=1}^{N} \frac{\partial \hat{y}_i}{\partial x_j}(x) \frac{\partial \hat{y}_i}{\partial x_k}(x) \tag{7.56}$$

and note that $a_{jj}(x) > 0$.

The Levenberg-Marquardt method modifies equation (7.55) by introducing the matrix $\hat{A}$ with entries

$$\hat{a}_{jj} = (1 + \gamma)\, a_{jj}$$
$$\hat{a}_{jk} = a_{jk} \quad j = k$$

where γ is some positive parameter. Equation (7.55) becomes

$$\hat{A}(x_0)(x_1 - x_0) = g \tag{7.57}$$

For large γ, the matrix $\hat{A}$ will become diagonally dominant. As γ approaches zero, equation (7.57) will turn into the Newton-Raphson method. The Levenberg-Marquardt method has the basic feature of varying γ to select the optimal characteristics of the iteration. The basic Levenberg-Marquardt algorithm is summarized:

Levenberg-Marquardt Method

1. Set $k = 0$. Choose an initial guess x_0, γ and a factor α.
2. Solve the linear system of equation (7.57) to obtain x_{k+1}.
3. If $f(x_{k+1}) > f(x_k)$, reject x_{k+1} as the new approximation, replace γ by $\alpha\gamma$, and repeat step 2.
4. If $f(x_{k+1}) < f(x_k)$, accept x_{k+1} as the new approximation, replace γ by γ/α, set $k = k + 1$, and repeat step 2.
5. Terminate the iteration when

$$\|x_{k+1} - x_k\| < \varepsilon$$

In the problem of estimating a nonlinear waveform by a series of functions, the minimization function is given by:

$$\text{minimize} f = \sum_{i=1}^{N} \left[\sum_{k=1}^{m} \left[a_k e^{(b_k t_i)} \cos(\omega_k t_i + \theta_k) \right] - y_i \right]^2 \tag{7.58}$$

where m is the number of desired modes of the approximating waveform, and $x = [a_1\ b_1\ \omega_1\ \theta_1\ \ldots a_m\ b_m\ \omega_m\ \theta_m]^T$.

As with all of the nonlinear iterative methods, the ability of the Levenberg-Marquardt method to converge to a solution depends on the choice of initial guess. In this case, it is wise to use the results of the matrix pencil or the Prony method to provide the initial values.

7.5.4 The Hilbert Transform

The shape of a signal that contains a rapidly oscillating component that veries slowly with time is called the "envelope." With the use of the Hilbert transform, the rapid oscillations can be removed from the signal to produce the representation of the envelope.

The Hilbert transform for $x(t)$ is:

$$X_H(t) = -\frac{1}{\pi t} x(t) = f(t)x(t) = F^{-1}\{F(j\omega)X(j\omega)\} \tag{7.59}$$

The Fourier transform of $(-\pi t)^{-1}$ is i sgn ω, which is $+i$ for positive ω and $-i$ for negative ω. The Hilbert transformation is equivalent to a filter in which the amplitudes of the spectral components are left unchanged, but their phases are altered by $\pi/2$, positively or negatively according to the sign of ω. The Hilbert transforms of even functions are odd and those of odd functions are even. The cosine component transforms into negative sine components and sine components transform into cosine components. A more in-depth explanation of the Hilbert transform can be in found in [22].

The impulse response function of a single-degree-of-freedom system is an exponential damped sinusoid. The Hilbert transform is used to calculate a new time signal from the original signal. Both the signals are combined to form the analytical signal

$$\bar{x(t)} = x(t) - iX_H(t) \tag{7.60}$$

The magnitude of the analytic signal is the envelope of the original time signal. When the envelope is plotted in the dB scale, the graph is a line. Then the slope of the line is related to the damping ratio. The impulse response function of a single-degree-of-freedom system can be described with the following equation:

$$x(t) = Ae^{-\xi\omega_n t} \sin\left(\omega_n \left(\sqrt{1-\xi^2}\right) t\right) \tag{7.61}$$

where ω_n is the natural frequency, ξ is the damping ratio, and A is the residue. The Hilbert transform of equation (7.61) is

$$X_H(t) = Ae^{-\xi\omega_n t} \cos\left(\omega_n \left(\sqrt{1-\xi^2}\right) t\right) \tag{7.62}$$

The analytic signal is

$$\bar{x(t)} = Ae^{-\xi\omega_n t}\left(\sin\left(\omega_n \left(\sqrt{1-\xi^2}\right) t\right) + i\cos\left(\omega_n \left(\sqrt{1-\xi^2}\right) t\right)\right) \tag{7.63}$$

The magnitude of the analytic signal eliminates the oscillatory component and gives the envelope:

$$\begin{aligned}|\bar{x(t)}| &= \sqrt{(Ae^{-\xi\omega_n t})^2\left(\sin^2\left(\omega_n\left(\sqrt{1-\xi^2}\right)t\right)+\cos^2\left(\omega_n\left(\sqrt{1-\xi^2}\right)t\right)\right)}\\ &= Ae^{-\xi\omega_n t}\end{aligned}$$

Taking the natural log of each side yields

$$\begin{aligned}\ln|\bar{x(t)}| &= \ln\left(Ae^{-\xi\omega_n t}\right)\\ &= \ln A - \xi\omega_n t\end{aligned}$$

This is the equation of a straight line in t. If the slope of the line is calculated, the estimate of the damping ratio becomes

$$\xi = -\frac{\text{slope}}{\omega_n} \tag{7.64}$$

The Hilbert transform provides an approach to estimate the number of modes in a waveform when the number of modes is not known beforehand.

7.5.5 Examples

The e ectiveness of these methods will be illustrated with several examples ranging from a simple three mode linear system to an actual power system oscillation.

Simple Example

The application of the methods will initially consider the waveform shown in Figure 7.3 which is generated from

$$x(t) = \sum_{i=1}^{3} a_i e^{b_i t}(\cos\omega_i t + \theta_i)$$

where

mode	a_i	b_i	ω_i	θ_i
1	1.0	-0.01	8.0	0.0
2	0.6	-0.03	17.0	π
3	0.5	0.04	4.7	$\pi/4$

If the problem were approached as if the system data were not known, applying the Hilbert transform would be the first step to estimate the number of modes in the system. The frequency response from the Hilbert transform is shown in Figure 7.4. From this figure, there are obviously three dominant modes at 4.7, 8.0 and 17 radians which agree with the given data very well.

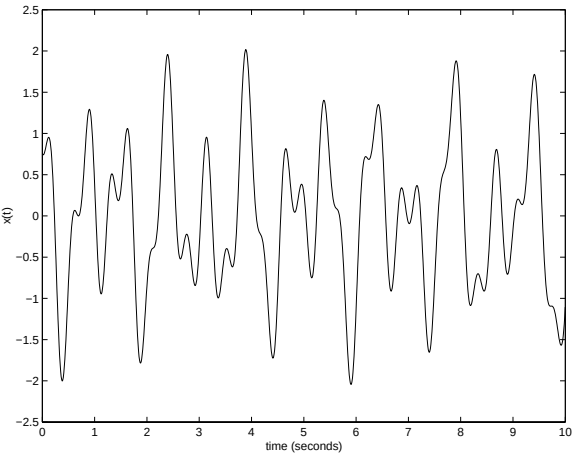


FIGURE 7.3
Three mode waveform

Furthermore, the amplitudes are $0.48, 0.9$ and 0.51 respectively, which also correspond relatively well. The damping at each resonance frequency can be determined if each natural frequency is isolated and the impulse response frequency is calculated for each mode. The slope of the envelope of impulse response can be used to estimate ξ for each frequency. After estimating the damping ratio of each mode, the eigenvalues can be calculated.

Each of the three methods described previously is used to estimate the signal parameters and to reconstruct the waveform.

Prony:

mode	a_i	b_i	ω_i	θ_i
1	0.9927	-0.0098	7.9874	0.0617
2	0.6009	-0.0304	17.0000	3.1402
3	0.5511	0.0217	4.6600	0.9969

Matrix Pencil

mode	a_i	b_i	ω_i	θ_i
1	1.0121	-0.0130	8.0000	0.0008
2	0.6162	-0.0361	16.9988	3.1449
3	0.5092	0.0364	4.6953	0.7989

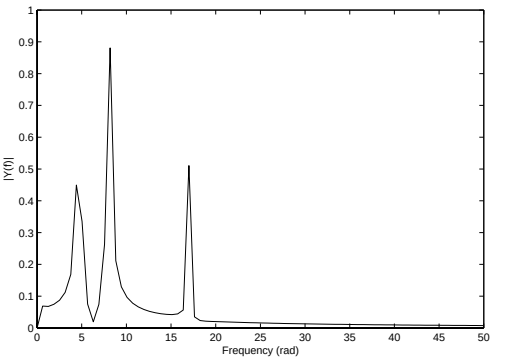


FIGURE 7.4
Frequency response of Figure 7.3

Levenberg-Marquardt

mode	a_i	b_i	ω_i	θ_i
1	1.0028	-0.0110	7.9998	0.0014
2	0.6010	-0.0305	16.9994	3.1426
3	0.5051	0.0378	4.6967	0.7989

The reconstruction error in each waveform is measured

$$\text{error} = \sum_{i=1}^{N} \left[\sum_{k=1}^{m} \left[a_k e^{(b_k t_i)} \cos(\omega_k t_i + \theta_k) \right] - y_i \right]^2 \tag{7.65}$$

and the errors for each method are:

Method	error
Matrix Pencil	0.1411
Levenberg-Marquardt	0.0373
Prony	3.9749

Not surprisingly, the Levenberg-Marquardt yielded the best results since it is an iterative method, whereas the other estimation methods are linear non-iterative methods.

Power System Example

In this example, the accuracy of the methods will be compared using the dynamic response of PSS/E simulation of a large Midwestern utility system shown in Figure 7.5. This simulation contains several hundred states comprising a wide range of responses. The number of dominant modes is not known.

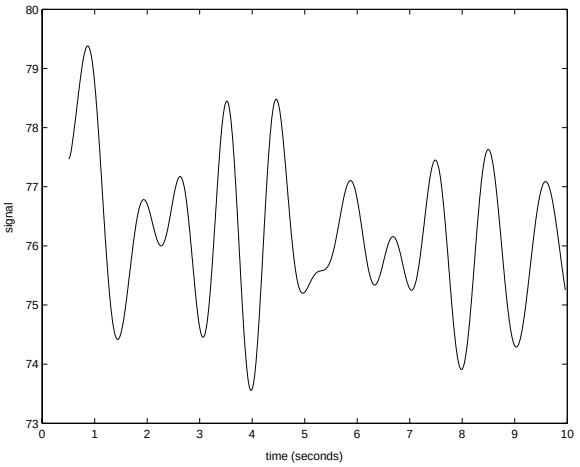


FIGURE 7.5
PSS/E waveform

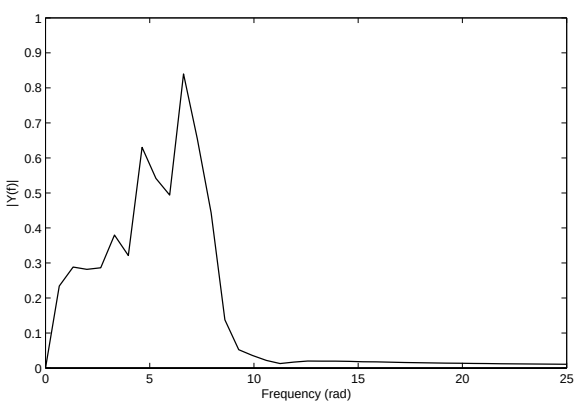
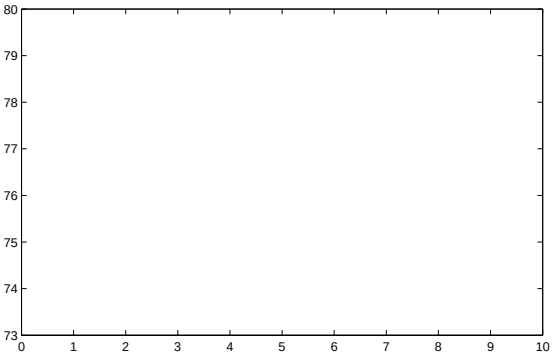


FIGURE 7.6
FFT of PSS/E waveform



Levenberg-Marquardt

mode	a_i	b_i	ω_i	θ_i
1	1.8604	-0.5297	3.6774	-1.3042
2	1.1953	-0.0578	4.8771	-1.3405
3	0.8164	0.0242	6.2904	-0.2537
4	1.9255	-0.2285	7.6294	2.1993
5	0.6527	-0.2114	8.3617	-1.5187

The error in each method as determined by equation (7.65) and the relative computation times are:

method	CPU	Error
Prony	0.068	228.16
Matrix Pencil	39.98	74.81
Levenberg-Marquardt	42.82*	59.74

* Depends on initial condition

Note that the Prony method is the most computationally e cient since it only requires two least squares solutions. The Matrix Pencil method is more computationally expensive since it requires a singular value decomposition of a relatively large matrix. It also requires an eigensolution, but since the matrix itself is relatively small (the size of the number of required modes), it is not overly burdensome. Not surprisingly, the Levenberg-Marquardt method is the most computationally expensive since it is an iterative method. Its computational burden is directly related to the initial guess: the better the initial guess, the faster the method converges. It is wise to choose the initial guess as the parameters obtained from either the Prony or the Matrix Pencil methods.

Similarly, the level of error in each method varies with the complexity of the method. The Levenberg-Marquardt method yields the best results, but with the greatest computational e ort. The Prony has the largest error, but this is o set by the relative speed of computation.

7.6 Power System Applications

7.6.1 Participation Factors

In the analysis of large scale power systems, it is sometimes desirable to have a measure of the impact that a particular state has on a selected system mode (or eigenvalue). In some cases, it is desirable to know whether a set of physical states has influence over an oscillatory mode such that control of

that component may mitigate the oscillations. Another use is to identify which system components contribute to an unstable mode. One tool for identifying which states significantly participate in a selected mode is the method of *participation factors* [57]. In large scale power systems, participation factors can also be used to identify inter-area oscillations versus those that persist only within localized regions (intra-area oscillations).

Participation factors provide a measure of the influence each dynamic state has on a given mode or eigenvalue. Consider a linear system

$$\dot{x} = Ax \tag{7.66}$$

The participation factor p_{ki} is a sensitivity measure of the i^{th} eigenvalue to the (k,k) diagonal entry of the system A matrix. This is defined as

$$p_{ki} \frac{\partial \lambda_i}{\partial a_{kk}} \tag{7.67}$$

where λ_i is the i^{th} eigenvalue and a_{kk} is the k^{th} diagonal entry of A. The participation factor p_{ki} relates the k^{th} state variable to the i^{th} eigenvalue. An equivalent, but more common expression for the participation factor is also defined as

$$p_{ki} = \frac{w_{ki} v_{ik}}{w_i^T v_i} \tag{7.68}$$

where w_{ki} and v_{ki} are the k^{th} entries of the left and right eigenvectors associated with λ_i. As with eigenvectors, participation factors are frequently normalized to unity, such that

$$\sum_{k=1}^{n} p_{ki} = 1 \tag{7.69}$$

When the participation factors are normalized, they provide a straightforward measure of the percent of impact each state has on a particular mode. Participation factors for complex eigenvalues (and eigenvectors) are defined in terms of magnitudes, rather than complex quantities. In the case of complex eigenvalues, the participation factors are defined

$$p_{ki} = \frac{|v_{ik}||w_{ki}|}{\sum_{i=1}^{n} |v_{ik}||w_{ki}|} \tag{7.70}$$

In some applications, it may be preferred to retain the complex nature of the participation factors to yield both phase and magnitude information [29].

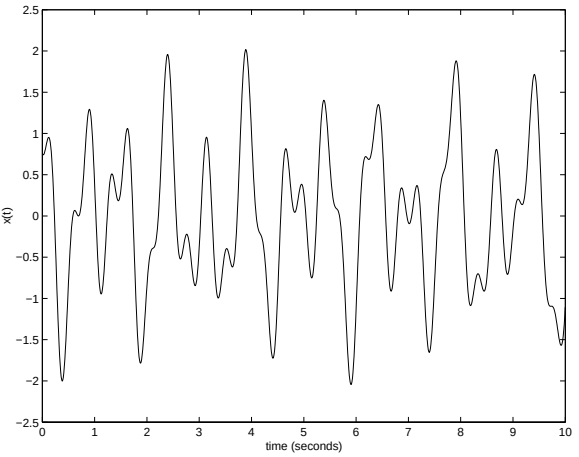


FIGURE 7.8
Waveform for Problem 3

7.7 Problems

1. Find the eigenvalues and eigenvectors of the following matrices

$$A_1 = \begin{bmatrix} 5 & 4 & 1 & 1 \\ 4 & 5 & 1 & 1 \\ 1 & 1 & 4 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 2 & 3 & 4 \\ 7 & -1 & 3 \\ 1 & -1 & 5 \end{bmatrix}$$

2. Find the eigenvalues of the follow matrix using the shifted-QR method.

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

3. Generate the waveform shown in Figure 7.8 on the interval $t \quad [0, 10]$ with a time step of 0.01 seconds

$$x(t) = \sum_{i=1}^{3} a_i e^{b_i t} (\cos c_i t + d_i)$$

where

mode	a_i	b_i	c_i	d_i
1	1.0	-0.01	8.0	0.0
2	0.6	-0.03	17.0	π
3	0.5	0.04	4.7	$\pi/4$

(a) Using 100 equidistant points on the interval $[0, 10]$, estimate the six system eigenvalues using Prony analysis. How do these compare with the actual eigenvalues?

(b) Using 100 equidistant points on the interval $[0, 10]$, estimate the six system eigenvalues using Levenberg-Marquardt. How do these eigenvalues compare with the actual eigenvalues? with those obtained from the Prony analysis?

(c) Using 100 equidistant points on the interval $[0, 10]$, estimate the six system eigenvalues using the matrix pencil method. How do these eigenvalues compare with the actual eigenvalues? with those obtained from the Prony analysis?

(d) Using all of the points, estimate the six system eigenvalues using Prony analysis. How do these compare with the actual eigenvalues?

(e) Using all of the points, estimate the six system eigenvalues using Levenberg-Marquardt. How do these compare with the actual eigenvalues?

(f) Using all of the points, estimate the six system eigenvalues using the matrix pencil method. How do these compare with the actual eigenvalues?

(g) Using all of the points, estimate the two dominant modes (two complex eigenvalue pairs) of the system response using the matrix pencil method. Substitute the estimated parameters into

$$x(t) = \sum_{i=1}^{2} a_i e^{b_i t} (\cos c_i t + d_i)$$

and plot this response versus the three mode response. Discuss the di erences and similarities.

References

[1] P. M. Anderson and A. A. Fouad, *Power System Control and Stability*. Ames, IA: Iowa State University Press, 1977.

[2] W. E. Arnoldi, "The principle of minimized iterations in the solution of the matrix eigenvalue problem," *Quart. Appl. Math.*, vol. 9, 1951.

[3] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 2nd ed. Wiley Press, 1993.

[4] L. T. Biegler, T. F. Colemam, A. R. Conn, and F. N. Santosa, *Large-scale Optimization with Applications - Part II: Optimal Design and Control.* New York: Springer-Verlag, Inc., 1997.

[5] K. E. Brenan, S. L. Campbell, L. R. Petzold, *Numerical Solution of Initial-Value Problems in Di erential-Algebraic Equations*. Philadelphia: Society for Industrial and Applied Mathematics, 1995.

[6] L. O. Chua and P. Lin, *Computer Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*. Englewood Cli s, New Jersey: Prentice-Hall, Inc., 1975.

[7] G. Dahlquist, A. Bjorck, *Numerical Methods*. Englewood Cli s, New Jersey: Prentice-Hall, Inc., 1974.

[8] G. Dantzig, *Linear Programming: Introduction*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997.

[9] R. Doraiswami and W. Liu, "Real-time estimation of the parameters of power system small signal oscillations," *IEEE Transactions on Power Systems*, vol. 8, no. 1, February 1993.

[10] H. W. Dommel and W. F. Tinney, "Optimal power flow solutions," *IEEE Transactions on Power Apparatus and Systems*, vol. 87, no. 10, pp. 1866-1874, October 1968.

[11] S. Eisenstate, M. Gursky, M. Schultz, and A. Sherman, "The Yale Sparse Matrix Package I: The symmetric codes," *International Journal of Numerical Methods Engineering*, vol. 18, 1982, pp. 1145-1151.

[12] O. I. Elgerd, *Electric Energy System Theory, An Introduction*. New York, New York: McGraw-Hill Book Company, 1982.

[13] J. Francis, "The QR Transformation: A unitary analogue to the LR Transformation," *Comp. Journal*, vol. 4, 1961.

[14] C. W. Gear, *Numerical Initial Value Problems in Ordinary Di erential Equations*, Englewood Cli s, New Jersey: Prentice-Hall, Inc., 1971.

[15] C. W. Gear, "The simultaneous numerical solution of di erential-algebraic equations," *IEEE Transactions on Circuit Theory*, vol. 18, pp. 89-95, 1971.

[16] C. W. Gear and L. R. Petzold, "ODE methods for the solution of differential/algebraic systems," *SIAM Journal of Numerical Analysis*, vol. 21, no. 4, pp. 716-728, August 1984.

[17] A. George and J. Liu, "A fast implementation of the minimum degree algorithm using quotient graphs," *ACM Transactions on Mathematical Software*, vol. 6, no. 3, September 1980, pp. 337-358.

[18] A. George and J. Liu, "The evoluation of the minimum degree ordering algorithm," *SIAM Review*, vol. 31, March 1989, pp. 1-19.

[19] H. Glavitsch and R. Bacher, "Optimal power flow algorithms," *Control and Dynamic Systems*, vol. 41, part 1, *Analysis and Control System Techniques for Electric Power Systems*, New York: Academic Press, 1991.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Baltimore: Johns Hopkins University Press, 1983.

[21] G. K. Gupta, C. W. Gear, and B. Leimkuhler, "Implementing linear multistep formulas for solving DAEs," Report no. UIUCDCS-R-85-1205, University of Illinois, Urbana, Illinois, April 1985.

[22] Stefan Hahn, *Hilbert Transforms in Signal Processing*, Boston: Artech House Publishers, 1996.

[23] J. F. Hauer, C. J. Demeure, and L. L. Scharf, "Initial results in Prony analysis of power system response signals," *IEEE Transactions on Power Systems*, vol. 5, no. 1, February 1990.

[24] Y. Hua and T. Sarkar, "Generalized Pencil-of-function method for extracting poles of an EM system from its transient response," *IEEE Transactions on Antennas and Propagation*, vol 37, no. 2, February 1989.

[25] M. Ilic and J. Zaborszky, *Dynamics and Control of Large Electric Power Systems*, New York: Wiley-Interscience, 2000.

[26] D. Kahaner, C. Moler, and S. Nash, *Numerical Methods and Software*, Englewood Cli s, NJ: Prentice-Hall, 1989.

[27] R. Kress, *Numerical Analysis*, New York: Springer-Verlag, 1998.

[28] V. N. Kublanovskaya, "On some algorithms for the solution of the complete eigenvalue problem," *USSR Comp. Math. Phys.*, vol. 3, pp. 637-657, 1961.

[29] P. Kundur, *Power System Stability and Control*. New York: McGraw-Hill, 1994.

[30] J. Liu, "Modification of the minimum-degree algorithm by multiple elimination," *ACM Transactions on Mathematical Software*, vol. 11, no. 2, June 1985, pp. 141-153.

[31] H. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Science*, vol. 3, 1957, pp. 255-269.

[32] A. Monticelli, "Fast decoupled load flow: Hypothesis, derivations, and testing," *IEEE Transactions on Power Systems*, vol. 5, no. 4, pp 1425-1431, 1990.

[33] J. Nanda, P. Bijwe, J. Henry, and V. Raju, "General purpose fast decoupled power flow," *IEEE Proceedings-C*, vol. 139, no. 2, March 1992.

[34] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, San Diego: Academic Press, Inc., 1970.

[35] A. F. Peterson, S. L. Ray, and R. Mittra, *Computational Methods for Electromagnetics*, New York: IEEE Press, 1997.

[36] M. J. Quinn, *Designing E cient Algorithms for Parallel Computers*, New York: McGraw-Hill Book Company, 1987.

[37] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, July 1986.

[38] O. R. Saavedra, A. Garcia, and A. Monticelli, "The representation of shunt elements in fast decoupled power flows," *IEEE Transactions on Power Systems*, vol. 9, no. 3, August 1994.

[39] J. Sanchez-Gasca and J. Chow, "Performance comparison of three identification methods for the analysis of electromechanical oscillations," *IEEE Transactions on Power Systems*, vol. 14, no. 3, August 1999.

[40] J. Sanchez-Gasca, K. Clark, N. Miller, H. Okamoto, A. Kurita, and J. Chow, "Identifying Linear Models from Time Domain Simulations," *IEEE Computer Applications in Power*, April 1997.

[41] T. Sarkar and O. Pereira, "Using the matrix pencil method to estimate the parameters of a sum of complex exponentials," *IEEE Antennas and Propagation*, vol. 37, no. 1, February 1995.

[42] P. W. Sauer and M. A. Pai, *Power System Dynamics and Stability*, Upper Saddle River, New Jersey: Prentice-Hall, 1998.

[43] J. Smith, F. Fatehi, S. Woods, J. Hauer, and D. Trudnowski, "Transfer function identification in power system applications," *IEEE Transactions on Power Systems*, vol. 8, no. 3, August 1993.

[44] G. Soderlind, "DASP3–A program for the numerical integration of partitioned sti ODEs and di erential/algebraic systems," Report TRITA-NA-8008, The Royal Institute of Technology, Stockholm, Sweden, 1980.

[45] D. C. Sorensen, "Implicitly restarted Arnoldi/Lanzcos methods for large scale eigenvalue calculations," in D. E. Keyes, A. Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms: Proceedings of an ICASE/LaRC Workshop, May 23-25, 1994, Hampton, VA*, Kluwer, 1995.

[46] D. C. Sorensen, "Implicit application of polynomial filters in a k-step Arnoldi method," *SIAM J. Mat. Anal. Appl.*, vol. 13, no. 1, 1992.

[47] P. A. Stark, *Introduction to Numerical Methods*, London, UK: The Macmillan Company, 1970.

[48] B. Stott and O. Alsac, "Fast decoupled load flow," *IEEE Transactions on Power Apparatus and Systems*, vol. 93, pp. 859-869, 1974.

[49] G. Strang, *Linear Algebra and Its Applications*, San Diego: Harcourt Brace Javanonich, 1988.

[50] W. Tinney and J. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorizations," *Proceedings of the IEEE*, vol. 55, no. 11, November 1967, pp. 1801-1809.

[51] L. Wang and A. Semlyen, "Application of sparse eigenvalue techniques to the small signal stability analysis of large power systems," *IEEE Transactions on Power Systems*, vol. 5, no. 2, May 1990.

[52] D. S. Watkins, *Fundamentals of Matrix Computations*. New York: John Wiley and Sons, 1991.

[53] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*. Oxford, England: Clarendon Press, 1965.

[54] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM Journal of Algebraic Discrete Methods*, vol. 2, 1981, pp. 77-79.

[55] T. Van Cutsem and C. Vournas, *Voltage Stability of Electric Power Systems*, Boston: Kluwer Academic Publishers, 1998.

[56] R. S. Varga, *Matrix Iterative Analysis*. Englewood Cli s, New Jersey: Prentice-Hall, Inc., 1962.

[57] G. C. Verghese, I. J. Perez-Arriaga, and F. C. Schweppe, "Selective modal analysis with applications to electric power systems," *IEEE Transactions on Power Systems*, vol. 101, pp. 3117-3134, Sept. 1982.

[58] W. Zangwill and C. Garcia, *Pathways to Solutions, Fixed Points, and Equilibria.* Englewood Cli s, New Jersey: Prentice-Hall, Inc., 1981.

In today's deregulated environment, the nation's electric power network