

Pramote Dechaumphai • N Wansophark

NUMERICAL METHODS IN SCIENCE AND ENGINEERING

Theories with MATLAB, Mathematica,
Fortran, C and Python Programs



Alpha Science

NUMERICAL METHODS IN SCIENCE AND ENGINEERING

Theories with MATLAB, Mathematica, Fortran, C and Python Programs

NUMERICAL METHODS IN SCIENCE AND ENGINEERING

Theories with MATLAB, Mathematica, Fortran, C and Python Programs

P. Dechaumpwwhai
N. Wansophark



**Alpha Science International Ltd.
Oxford, U.K.**

Numerical Methods in Science and Engineering

Theories with MATLAB, Mathematica, Fortran, C and Python Programs
388 pgs.

P. Dechaumphai

N. Wansopha

Mechanical Engineering Department
Chulalongkorn University
Payathai Road, Pathumwan
Bangkok 10330, Thailand

Copyright © 2020

ALPHA SCIENCE INTERNATIONAL LTD.

7200 The Quorum, Oxford Business Park North
Garsington Road, Oxford OX4 2JZ, U.K.

www.alphasci.com

ISBN 978-1-78332-554-2

E-ISBN 978-1-78332-579-5

Printed from the camera-ready copy provided by the Authors.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without prior written permission of the publisher.

Preface

The book, *Numerical Methods in Science and Engineering: Theories with MATLAB, Mathematica, Fortran, C and Python Programs*, is written in a clear, easy-to-understand manner on theories and the use of the numerical methods. Topics and materials of the methods in this book were taught at George Washington University, NASA Langley Research Center campus while the first author was a NASA aerospace engineer. Such materials were also taught at Old Dominion University, Norfolk, Virginia, and have been currently taught at Chulalongkorn University. By teaching and performing research on the numerical methods for the past 30 years, the materials in this book have been improved and updated continuously. The main objective of this book is to present the numerical methods in their simplest form so that engineers and scientists can understand them easily and quickly.

The book contains 8 chapters which are essential in the study of the numerical methods. The materials in these chapters are suitable to be used in both the undergraduate and graduate levels. The first chapter introduces the methods and the need to study them for solving practical engineering problems today. The chapter also explains different types of numerical errors and the use of hardware and software. The second chapter explains several methods for finding roots from a single nonlinear equation. The methods are extended to find roots from a set of nonlinear equations. Popular methods for finding roots by solving a set of linear simultaneous equations are presented in chapter 4. These methods are classified into two groups of the direct and iterative techniques. Their detailed computational procedures including advantages and disadvantages are presented. Interpolation and extrapolation methods for finding an appropriate function to represent a set of data are presented in chapter 4. Chapter 5 explains several least-squares regression methods to provide a function that best fit a set of data. Many types of functions to best fit sets of linear and nonlinear data are presented. Numerical integration and differentiation methods are explained in chapter 6. Basic and popular integration methods that are employed in commercial software for analyzing practical engineering problems are explained. Chapter 7 presents several methods for solving the ordinary differential equations. The methods can be used to analyze the first- and higher-order ordinary differential equations. Methods for solving partial differential equations are presented in chapter 8. The finite difference methods for analyzing the elliptic, parabolic and hyperbolic differential equations are explained in details. For all the methods presented in these chapters, listings of the corresponding computer programs are provided. These computer programs are written in MATLAB, Mathematica, Fortran, C, Pascal and Python so that readers can select the preferred computer language. These programs can be downloaded from the book website:

<http://bit.ly/2WCRIExNumCodes>

The computational procedures in these computer programs follow the theories presented in the text. They are easy to modify for solving a large number of problems at the end of chapters. Readers are encouraged to practice with these programs to appreciate the ability of numerical methods that are embedded in commercial software today.

The first author would like to thank his former Professor, Dr. Earl A. Thornton, and his supervisor, Dr. Allan R. Wieting of the Aerothermal Loads Branch at NASA Langley Research Center. He expresses his appreciation to the students at NASA Langley Research Center, Old Dominion University and Chulalongkorn University who took his courses on the numerical methods and helped him to improve the presentation of materials in this book. The authors wish to thank Mr. Sascha J. Mehra, the Director and the staff of Alpha Science International Ltd. for their advice and cooperation. The authors appreciate Dr. Edward Dechaumpahi and Mrs. Anna D. McDermott for proofreading the book. The authors would like to thank their wives Mrs. Yupa Dechaumphai and Patcharin Wansophark for the understanding and support in writing this book.

Pramote Dechaumphai

Niphon Wansophark

Contents

<i>Preface</i>	v
1. First Step to Numerical Methods	1
1.1 Introduction	1
1.2 What are the Numerical Methods?	3
1.3 Need for Studying Numerical Methods	3
1.4 Computer Hardware and Software	8
1.5 Errors	11
1.6 Closure	13
Exercises	13
2. Root of Equations	19
2.1 Introduction	19
2.2 Graphical Method	20
2.3 Bisection Method	22
2.4 False-Position Method	25
2.5 One-Point Iteration Method	28
2.6 Newton-Raphson Method	32
2.7 Secant Method	37
2.8 MATLAB Functions for Finding Root of Equation	38
2.9 Roots of System of Non-linear Equations	41
2.9.1 Direct iteration method	42
2.9.2 Newton-Raphson iteration method	43
2.10 Closure	46
Exercises	46
3. System of Linear Equations	53
3.1 Introduction	53
3.2 Cramer's Rule	55
3.3 Gauss Elimination Method	57
3.4 Problems of Gauss Elimination Method	62
3.4.1 Division by zero	62
3.4.2 Round-off error	62
3.4.3 Ill-conditioned system	63

3.5 Improved Gauss Elimination Method	63
3.5.1 Pivoting	64
3.5.2 Scaling	65
3.5.3 Tridiagonal system	66
3.6 Gauss-Jordan Method	68
3.7 Matrix Inversion Method	70
3.8 Solving System of Linear Equations by MATLAB	71
3.9 LU Decomposition Method	72
3.10 MATLAB Function for LU Decomposition	77
3.11 Cholesky Decomposition Method	78
3.12 MATLAB Function for Cholesky Decomposition	81
3.13 Jacobi Iteration Method	82
3.14 Gauss-Seidel Iteration Method	85
3.15 Successive Over-relaxation Method	87
3.16 Conjugate Gradient Method	88
3.17 Closure	101
Exercises	101
4. Interpolation and Extrapolation	111
4.1 Introduction	111
4.2 Newton's Divided Differences	112
4.2.1 Linear interpolation	112
4.2.2 Quadratic interpolation	113
4.2.3 n^{th} -order Polynomial interpolation	115
4.3 Lagrange Interpolating Polynomials	118
4.3.1 Linear interpolation	118
4.3.2 Quadratic interpolation	120
4.3.3 Polynomial interpolation	122
4.4 Spline Interpolations	124
4.4.1 Linear spline	125
4.4.2 Quadratic spline	126
4.4.3 Cubic spline	128
4.5 MATLAB Functions for Interpolations	132
4.6 Extrapolation	134
4.7 Closure	135
Exercises	136
5. Least-Squares Regression	141
5.1 Introduction	141
5.2 Linear Regression	142
5.3 Linear Regression for Nonlinear Data	146
5.4 Polynomial Regression	150
5.5 MATLAB Functions for Least-Squares Regression	154

5.6	Multiple Regression	156
5.6.1	Linear	156
5.6.2	Polynomial	162
5.7	Closure	164
	Exercises	164
6.	Numerical Integration and Differentiation	173
6.1	Introduction	173
6.2	Trapezoidal Rule	175
6.3	Composite Trapezoidal Rule	180
6.4	Simpson's Rule	184
6.5	Composite Simpson's Rule	186
6.6	Newton-Cotes Formulas	188
6.7	Romberg Integration	192
6.8	Gauss Integration	197
6.9	Multiple Integration	205
6.10	MATLAB Commands for Integration	208
6.11	Differentiation	210
6.12	MATLAB Commands for Differentiation	216
6.13	Closure	217
	Exercises	217
7.	Ordinary Differential Equations	227
7.1	Introduction	227
7.2	Euler's Method	230
7.3	Heun's Method	234
7.4	Modified Euler's Method	237
7.5	Runge-Kutta Method	239
7.5.1	Second-order	240
7.5.2	Third-order	242
7.5.3	Fourth-order	243
7.6	System of Equations	246
7.7	MATLAB Commands	250
7.8	Multistep Methods	252
7.8.1	Non-self-starting Heun's method	253
7.8.2	Adams-Basforth method	255
7.8.3	Adams-Moulton method	258
7.9	Closure	260
	Exercises	260
8.	Partial Differential Equations	269
8.1	Introduction	269

8.1.1	Definitions	269
8.1.2	Types of equations	270
8.1.3	Boundary and initial conditions	272
8.2	Elliptic Equation	273
8.2.1	Differential equation	273
8.2.2	Computational procedures	274
8.2.3	Example	277
8.3	Parabolic Equation	283
8.3.1	Differential equation	283
8.3.2	Explicit method	284
8.3.3	Implicit method	289
8.3.4	Crank-Nicolson method	292
8.4	Hyperbolic Equation	297
8.4.1	Differential equation	297
8.4.2	Computational procedures	299
8.4.3	Example	301
8.5	Closure	305
	Exercises	305

Bibliography	317
---------------------	------------

Appendix A Matrices	319	
A.1	Definitions	319
A.2	Matrix Addition and Subtraction	321
A.3	Matrix Multiplication	321
A.4	Matrix Transpose	322
A.5	Matrix Inverse	322
A.6	Matrix Partitioning	323
A.7	Calculus of Matrices	323

Appendix B MATLAB Fundamentals	325
---------------------------------------	------------

B.1	Introduction	325
B.2	MATLAB Environment	325
B.2.1	Command Window	326
B.2.2	Command History Window	328
B.2.3	Edit/debug Window	328
B.2.4	Figure Window	329
B.2.5	Workspace Window	330
B.2.6	Help Window	330
B.3	Variables in MATLAB	331
B.3.1	Scalar variable	331
B.3.2	Vector variable	331
B.3.3	Use of colon symbol	333
B.3.4	Displaying data	333
B.3.5	Use of long commands	334

B.4 Mathematical Operations	334
B.5 Built-In Functions	336
B.6 Plotting Graphs	338
B.7 Programming	343
B.7.1 Script file	343
B.7.2 Function file	344
B.7.3 Input and output commands	344
B.7.4 Read and write data file	347
B.7.5 Programming commands	349
B.7.5.1 Decision commands	349
B.7.5.2 Iteration commands	352
Appendix C Derivation of Fourth-Order Runge-Kutta Formula	355
Appendix D Mathematica Commands	359
Index	371

Chapter

1

First Step to Numerical Methods

1.1 Introduction

Solving problems in sciences and engineering today requires knowledge in numerical methods. The methods are based on the use of mathematics, computational procedures, computer software and hardware for analyzing practical problems that normally have complex geometry. For examples, the finite volume method may be used to determine the flow behavior surrounding a moving vehicle. The computed pressure can be used in the modification of the vehicle body in order to reduce the drag force. The finite element method may be used to analyze the strength of the vehicle body structure to reduce its damage during a collision. The method can be also applied to analyze the temperature and associated thermal stress that occur on the automobile engine during running. Understanding such phenomena from the solutions by using the numerical methods helps designers to significantly reduce the time and cost for designing new products.

Designing a vehicle body structure with maximum strength or developing a new engine with reduced thermal stress was not possible in the past by using classical mathematics for exact solutions. The exact solutions can not be obtained because both of the geometries and boundary conditions of these problems are normally complex. The numerical methods, such as the finite element, finite volume and finite difference methods, are being used to obtain approximate solutions. These methods play a very important role in engineering analysis and design today. Figure 1.1 shows a finite difference mesh for an analysis of flow field surrounding a fighter jet. For problems that have complex geometries, the finite element method is often applied to obtain solutions. Figure 1.2 shows a finite element mesh of a vehicle body structure during its collision.

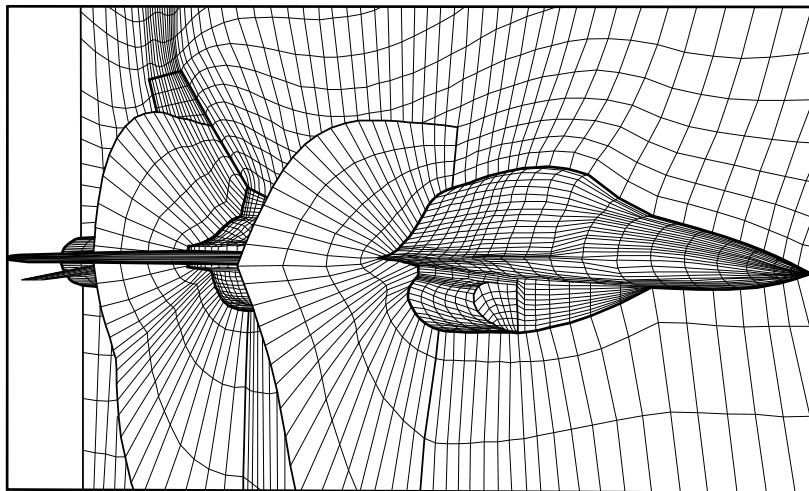


Figure 1.1 A finite difference mesh for an analysis of flow field surrounding a fighter jet.

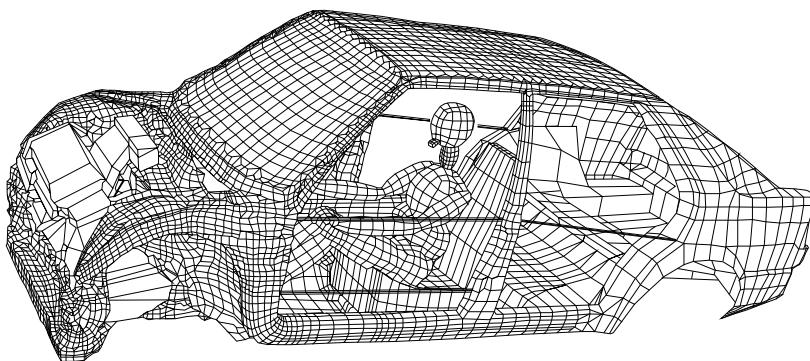


Figure 1.2 A finite element mesh of a vehicle body structure during collision.

These numerical methods significantly help designers and analysts to understand the problem behaviors in order to improve their designs. However, designers and analysts must understand the numerical methods prior to using them. These numerical methods are not difficult to understand and currently being taught in most of the sciences and engineering schools.

1.2 What are the Numerical Methods?

The numerical methods are techniques for obtaining approximate solutions. The methods consist of computational procedures that can be followed in the form of patterns. Generally, the procedures consist of arithmetic operations. These operations are simply the addition, subtraction, multiplication and division. A computational procedure can be performed by using a calculator for a small problem. The same procedure needs help from a computer program to provide solutions for a larger problem. The numerical methods are thus similar to the recipes in a cookbook. Users can follow the recipes or procedures to generate solutions. At present, many commercial software have been developed and used widely. These software contain numerical methods with some specific procedures to provide approximate solutions. Thus, it is very important for users to understand the computational procedures that are built inside the software prior to using them.

With the explanation above, it is clear that the concepts of the numerical methods are not new. Many numerical methods have been developed long time ago. They have not been used effectively, however, due to the lack of digital computers. At present, high efficient computer notebooks and laptops are available at low cost. The methods are thus popular and widely used by students and designers for obtaining solutions that are not possible in the past by using classical mathematics.

However, there are many practical applications today that can not be solved effectively by the numerical methods and current computers. For an example, analysis of flow field surrounding an aerospace vehicle that travels many times faster than the sound speed. The flow field is very complex and requires a computational mesh with a large number of grid points. Such problem is still a challenging problem today because it needs a very large computer memory as well as a substantial computational time.

1.3 Need for Studying Numerical Methods

In this section, an example is presented to demonstrate advantages of using a numerical method as compared to the classical mathematics method for solving a typical problem. In addition to the presentation of the solution procedure, the example highlights the importance in understanding the physical meaning of the problem. Understanding physical meanings before solving a problem is important because a proper numerical method can be selected to provide a solution with high accuracy. Understanding physical meaning of a problem is also very helpful, especially when analyzing more complex problems. Such understanding is required mainly because:

- (a) there is no single numerical method that can provide solution to every problem,
- (b) error of the solution always occurs from a numerical method, and
- (c) there is no single numerical method that is the best for all problems.

With the above statements, it is very important to understand the computational procedures of the numerical methods clearly. Understanding the computational procedure can lead to a more accurate solution. In addition, it also provides confidence for the user on the obtained solution, especially when solving a complex problem.

In order to demonstrate that the numerical methods are not difficult to understand as compared to the classical mathematics, a following example is studied.

Example 1.1 Apply the Newton's second law to develop a governing differential equation for approximately determining the space shuttle velocity during its descending as shown in Fig. 1.3. Derive the exact solution and develop a numerical procedure for obtaining an approximate solution of the velocity. Compare the approximate velocity solution with the exact solution.

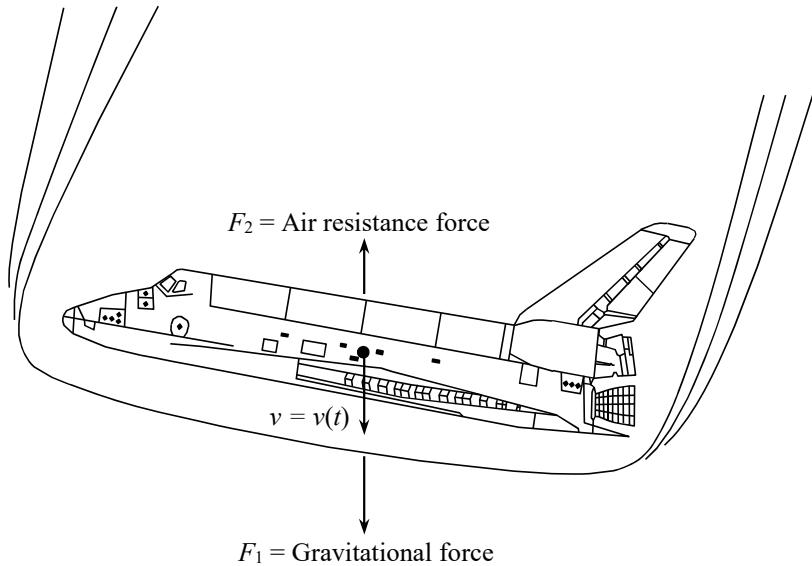


Figure 1.3 Forces on space shuttle during descending.

Solution From the Newton's second law,

$$F = ma \quad (1.1)$$

where F represents the net force which is

$$F = F_1 - F_2 \quad (1.2)$$

In Eq. (1.2), F_1 is the gravitational force defined by

$$F_1 = mg \quad (1.3)$$

where m is the mass of the shuttle and g is the gravitational acceleration constant. If the air resistance force F_2 is assumed to vary linearly with the velocity v , then

$$F_2 = cv \quad (1.4)$$

where c represents the drag coefficient which depends on the shuttle geometry. By substituting Eqs. (1.3) and (1.4) into Eq. (1.2), Eq. (1.1) becomes

$$mg - cv = ma$$

Because acceleration is the rate of change of the velocity, then

$$mg - cv = m \frac{dv}{dt}$$

The above equation is a linear ordinary differential equation that can be written as,

$$\frac{dv}{dt} + \frac{c}{m}v = g \quad (1.5)$$

where the unknown is the velocity v which depends on time t .

- The velocity v can be determined by solving the differential Eq. (1.5) using
- a mathematical method for an exact solution, or
 - a numerical method for an approximate solution.

The exact solution can be derived by using the method of separation of variables. Equation (1.5) is rewritten as

$$\frac{dv}{dt} = g - \frac{c}{m}v$$

i.e., separate the independent variable t and the dependent variable v so that they are on opposite side of the equation. Integration is then performed on both sides

$$\int \frac{dv}{g - \frac{c}{m}v} = \int dt \quad (1.6)$$

to yield

$$-\frac{m}{c} \ln\left(g - \frac{c}{m}v\right) = t + A \quad (1.7)$$

where A is the integrating constant that can be determined from the given initial condition. For example, if the velocity $v = 0$ at time $t = 0$, then Eq. (1.7) gives

$$A = -\frac{m}{c} \ln g \quad (1.8)$$

By substituting A from Eq. (1.8) into Eq. (1.7),

$$\begin{aligned} -\frac{m}{c} \ln\left(g - \frac{c}{m}v\right) &= t - \frac{m}{c} \ln g \\ \frac{m}{c} \ln\left(g - \frac{c}{m}v\right) - \frac{m}{c} \ln g &= -t \\ \ln\left(1 - \frac{c}{mg}v\right) &= -\frac{c}{m}t \\ 1 - \frac{c}{mg}v &= e^{-\frac{c}{m}t} \\ \frac{c}{mg}v &= 1 - e^{-\frac{c}{m}t} \\ v &= \frac{mg}{c} \left(1 - e^{-\frac{c}{m}t}\right) \end{aligned} \quad (1.9)$$

The exact velocity solution as shown in Eq. (1.9) indicates that the velocity is zero at time $t = 0$ as given by the initial condition. The velocity then increases with time and reaches a constant value as the time approaches infinity,

$$v(t \rightarrow \infty) = \frac{mg}{c} \quad (1.10)$$

At such condition, the air resistance force F_2 and the gravitational force F_1 are equal.

By assigning the following data:

$$\begin{array}{lll} \text{mass of space shuttle} & m = 90,000 \text{ kg} \\ \text{drag coefficient} & c = 450 \text{ kg/sec} \\ \text{gravitational acceleration constant} & g = 9.8 \text{ m/sec}^2 \end{array} \quad (1.11)$$

then, the expression for the shuttle velocity in Eq. (1.9) is,

$$v(t) = 1,960(1 - e^{-0.005t}) \quad (1.12)$$

The values of the velocity at every 30 seconds are shown in Table 1.1.

Table 1.1 Exact shuttle velocities at every 30 seconds according to Eq. (1.12).

Time t , sec	Velocity v , m/sec
0	0
30	273
60	508
90	710
120	884
150	1,034
180	1,163
⋮	⋮
⋮	⋮
∞	1,960

In stead of finding the exact solution from the differential equation, an approximate solution can be derived. The rate of change of the velocity dv/dt in Eq. (1.5) can be approximated by considering the plot of the velocity versus time in Fig. 1.4. The rate of change of the velocity dv/dt , which is the slope of the velocity v with respect to time t at point A , may be approximated by

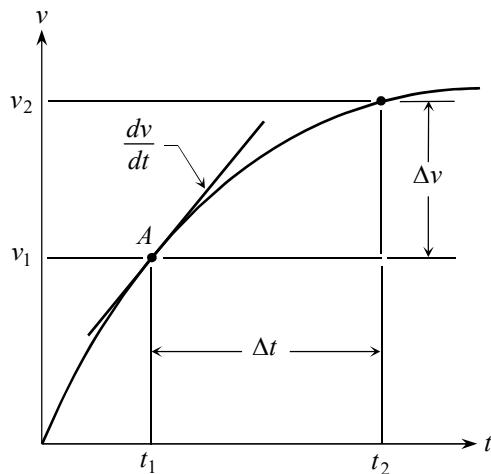


Figure 1.4 Plot of the velocity with time.

$$\frac{\Delta v}{\Delta t} = \frac{v_2 - v_1}{t_2 - t_1} \quad (1.13)$$

Such approximation $\Delta v/\Delta t$ becomes dv/dt as $\Delta t \rightarrow 0$. This means the approximation of Eq. (1.13) is accurate if the time step Δt is small. Using such approximation, Eq. (1.5) may be written as

$$\frac{v_2 - v_1}{\Delta t} + \frac{c}{m} v_1 = g$$

The above equation can be written in a more general form for $i = 1, 2, 3, \dots$ as

$$\frac{v_{i+1} - v_i}{\Delta t} + \frac{c}{m} v_i = g \quad (1.14)$$

Or,

$$v_{i+1} = v_i + \Delta t \left(g - \frac{c}{m} v_i \right) \quad (1.15)$$

With the values of the shuttle mass m , the drag coefficient c and the gravitational acceleration constant g as shown in Eq. (1.11), Eq. (1.15) becomes,

$$v_{i+1} = v_i + \Delta t (9.8 - 0.005v_i) \quad (1.16)$$

Equation (1.16) suggests that if the time step Δt and the velocity at step i are known, the velocity at step $i+1$ can be determined directly. By using the time step $\Delta t = 30$ s and the initial velocity of zero, table 1.2 shows the approximate solution of Eq. (1.16) as compared to the exact solution of Eq. (1.12).

Table 1.2 Comparative exact and approximate velocity solutions by using the time step $\Delta t = 30$ seconds.

<i>i</i>	<i>i</i> + 1	<i>t</i> , sec	Approximate solution Eq. (1.16)	Exact solution Eq. (1.12)
0	1	30	294	273
1	2	60	544	508
2	3	90	756	710
3	4	120	937	884
4	5	150	1,090	1,034
5	6	180	1,221	1,163
⋮	⋮	⋮	⋮	⋮
49	50	1,500	1,959	1,959

Both of the exact and approximate solutions are compared by the plot as shown in Fig. 1.5. With the time step $\Delta t = 30$ s, the approximate solution is in good agreement with the exact solution. From the derivation of the exact solution, the computational procedure for generating the approximate solution and the comparison of both the solutions in Fig. 1.5, the following details are observed:

(a) the approximate solution in Eq. (1.15) obtained from the numerical method can be derived easily as compared to the derivation of the exact solution,

(b) the approximate solution in Eq. (1.15) can be computed easily by developing a short computer program,

(c) the computer program can be executed by using a small time step to produce a more accurate solution,

(d) If the air resistance force F_2 does not vary linearly with the velocity, e.g., if it varies with the velocity in the form,

$$F_2 = cv^4 \quad (1.17)$$

then, the governing differential equation becomes nonlinear, i.e.,

$$\frac{dv}{dt} + \frac{c}{m}v^4 = g \quad (1.18)$$

In this case, the exact solution is difficult to find. However, the approximate solution can be determined by using the same procedure as explained earlier.

From the above reasons, the numerical methods are popular and being used by scientists and engineers for solving problems. Practical problems can be analyzed effectively if users have some backgrounds on both the computer hardware and software as explained in the following section.

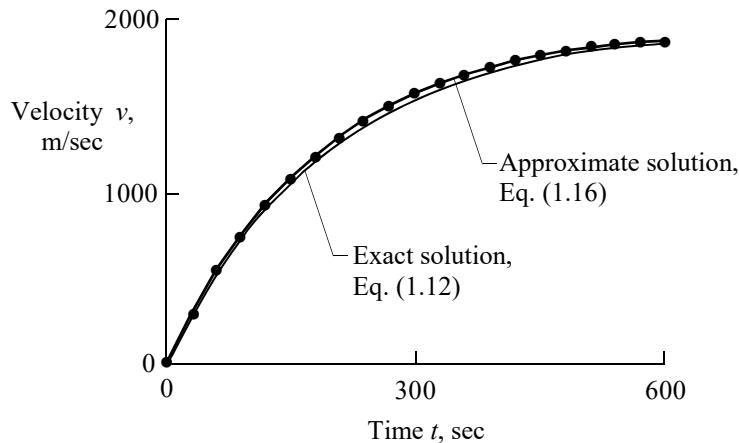


Figure 1.5 Comparison between the exact and approximate solutions for the shuttle velocity with respect to time.

1.4 Computer Hardware and Software

Studying a numerical method requires two basic ingredients: (1) the understanding of the theory and computational procedure of the method, and (2) the experience in using computers on both the hardware and software. The following chapters on this book explain theories of the numerical methods and their computational procedures. The computational procedures are translated into computer

programs in different languages of Fortran, MATLAB, Mathematica, Python and C. These computer programs can be executed on different types of computer notebooks and laptops. Students are encouraged to study theories and their computational procedures prior to using the computer programs.

During the past decades, computer hardware has been improved significantly. A large size problem can be analyzed quickly and conveniently on a computer notebook or laptop today. In the past, such a problem must be solved on a mainframe computer or a workstation. A mainframe computer was operated in an air-conditioned room with controlled temperature. The computer also required an operator for maintenance and service during its operation. At present, computer notebooks and laptops are widely used to analyze many practical problems. A large size problem may be solved by connecting these computers together in parallel. For a very large size problem, a supercomputer is normally used. These supercomputers are expensive and employed only by some big companies or government agencies.

It should be kept in mind that, like a calculator, these computers are used to perform basic operations of addition, subtraction, multiplication and division. But they can perform such operations at a very high efficiency, especially if they were instructed by computer software. A software for performing computational procedure of a numerical method may be written by using different computer languages. Popular languages are such as Fortran, Pascal, C, Java and Basic in the past. Nowadays, many software packages that contain different numerical methods have been developed and used widely. Examples of these software packages are MATLAB, Mathematica and Maple. Because these software packages are handy and easy to use, they are accepted by students to solve small size problems or employ for their projects. For the large size problems that occur in engineering applications, special software are employed. These special software, sometimes known as Computer-Aided Engineering (CAE) software, are very expensive. The software are being used for analysis and design of new products in automotive, electronics, medical and aerospace industry. They were developed by experience programmers who understand both the theories and computational procedures very well.

As a student who needs to learn the numerical methods, a simple computer language should be selected. The most important aspect is that the methods and their computational procedures must be understood clearly prior to developing any computer program. Such understanding will provide basis for solving more complex problems that occur in practical applications.

Few key ingredients are needed during developing a computer program:

(a) *Understanding computer commands.* There are few computer commands that are frequently used during developing or running a computer program. These commands are such as copying, deleting, editing, reading files, etc.

(b) *Knowing how to edit a file.* Once a computer program is created as a file, editing and correcting it are needed before it can be used. File editing is quite simple today. Many computer systems allow users to edit the files conveniently on the monitor screen.

(c) *Understanding computer languages.* Understanding a computer language clearly is a must for developing a good computer program. The developed program should be simple and provides high computational efficiency at the same time.

To help readers on computer programming, this book contains a number of computer programs that correspond to the computational procedures of the presented numerical methods. The programs are written in Fortran, MATLAB, Mathematica, Python and C languages. An example of a computer program is shown in the example below.

Example 1.2 The approximate solution of the space shuttle velocity as shown in Eq. (1.16) and Table 1.2 can be obtained by using the computer programs written in MATLAB in Fig. 1.6.

```
% Program Shuttle
nsteps = 20; dt = 30;
% Initial condition:
t(1) = 0.; v(1) = 0;
disp(' Time Velocity')
for i=1:nsteps
    v(i+1) = v(i) + dt*(9.8 - .005*v(i));
    t(i+1) = t(i) + dt;
    fprintf('%8.0f %8.0f\n', t, v)
end
plot(t,v,'-ok'), hold on
% Exact solution:
te = 0:30:600;
ve = 1960*(1.-exp(-.005*te));
plot(te,ve,'-k')
```

Figure 1.6 Computer program for determining the shuttle velocity according to Eq. (1.16).

Example 1.3 The sine and cosine functions can be written in the form of infinite series as

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots \quad (1.19)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots \quad (1.20)$$

where x is the angle in radians. Develop a computer program to compute values of both functions from 0 to 180 degrees with an increment of every 10 degrees.

The computer programs in MATLAB for determining both sine and cosine functions are shown in Fig. 1.7. Solutions of the sine and cosine functions obtained from these programs are presented in Table 1.3.

```
% Program SinCos
% Computing sin and cosine functions
% for the angles from 0 to 180 degrees
% with increment at every 10 degrees.
deg = 0.; del = 10.;
fprintf('%10s %16s %16s\n', ...
        'Degrees','Sin','Cos');
for ideg = 1:10
    x = pi*deg/180.;
    sums = x; sumc = 1.;
    terms = x; termc = 1.;
    sign = -1.;
    for n = 1:100
        ms = 2*n + 1; mc = 2*n;
        terms = terms*x*x/(ms*(ms-1));
        termc = termc*x*x/(mc*(mc-1));
        sums = sums + sign*terms;
        sumc = sumc + sign*termc;
        sign = -sign;
    end
    fprintf('%10.0f %16.6f %16.6f\n', ...
            deg, sums, sumc);
    deg = deg + del;
end
```

Figure 1.7 Computer programs for determining sine and cosine functions.

Table 1.3 Values of sine and cosine functions computed from the infinite series by using the computer programs in Fig. 1.7.

Degrees	Sin	Cos
0.	.000000	1.000000
10.	.173648	.984808
20.	.342020	.939693
30.	.500000	.866025
40.	.642788	.766044
50.	.766044	.642788
60.	.866025	.500000
70.	.939693	.342020
80.	.984808	.173648
90.	.000000	.000000
100.	.984808	-.173648
110.	.939693	-.342020
120.	.866025	-.500000
130.	.766044	-.642788
140.	.642788	-.766044
150.	.500000	-.866025
160.	.342020	-.939693
170.	.173648	-.984808
180.	.000000	-1.000000

1.5 Errors

Example 1.1 shows a computational procedure for determining the space shuttle velocity by using a simple numerical method. The numerical method requires less effort to provide a solution as compared to the use of classical mathematics. However, the approximated solution obtained from the numerical method has error which occurs from the use of a large time step. The error can be reduced by using a smaller time step, but the problem needs more computational time. For practical problems with a large number of unknowns, using a small time step may be prohibited because too large computational time is required. A proper time step, thus, must be decided prior to analyzing a large size problem.

In addition to the error that occurs from the use of time step as explained above, there are other types of errors that may arise from different sources. These errors are:

(a) *Modeling error.* Analysis of a problem normally starts from the discretization of the problem domain into small chunks of elements. These elements are connected at grid points where the unknowns are located and determined. The error is introduced because the continuum model is transformed into a discrete model. A large error occurs if the discrete model contains only few elements. The discrete model using small elements will produce a solution with less error. However, the model with small elements contains a large number of grid points and unknowns. High computer memory and computational time are required thus for the solution.

(b) *Propagation of error.* An error may propagate from one solution to another. As in the example for determination of the shuttle velocity, the error that occurs from the computed solution at 30 seconds can propagate to produce an additional error in the computed solution at 60 seconds. Or in the example of the car collision as shown in Fig. 1.2, the error that occurs in the deformed structure solution at an early time can propagate to produce more error of the structure solution at the later time. The propagation of error must be realized, especially when analyzing a practical problem containing a large number of unknowns.

(c) *Error from data.* Uncertain data produces error in the solution. As in Ex. 1.1, the space shuttle mass may not be exactly 90,000 kg and thus the computed solution is altered from the actual situation. Or in the example of the car collision in Ex. 1.2, the actual material stiffness may be different from that used in the computation. However, in many cases, actual data may not be available. Analysts must make judgment and be very careful prior to selecting proper data in order to obtain accurate and reasonable solutions.

(d) *Blunder error.* Careless programmers can produce serious error in the computed solution. Such error may come from typing data incorrectly, writing a computer program without checking it thoroughly, etc. The error may come from incorrect statements in computer programs. A new computer program must be inspected or debugged comprehensively to assure that it will not produce such error.

(e) *Truncation error.* The truncation error occurs when some terms are omitted or excluded from the equations during computation. For example, higher-order terms are omitted in the computation of the infinite series for the sine and cosine functions in Ex. 1.3. Truncation error always occurs in the computation of infinite series that are arisen from exact solutions of academic type problems.

(f) *Round-off error.* The round-off error arises from the use of computers that have limited capability in storing values. For example, the value of π that consists of 25 significant figures is

$$\pi = 3.141592653589793238462643 \quad (1.21)$$

Such the value of π with 25 significant figures can be stored on any computer today. Few decades ago, a typical computer may store only 10 significant figures, so that the value 3.141592653 was used for π in the computation.

The number of significant figures is used to indicate the accuracy of solution obtained from a numerical method. If the value of π is given by 3.14159, it has six significant figures. Frequently, most of the values used in numerical methods are in the floating point format. The π value of 3.14159 is written in the floating point format that has six significant figures as 0.314159×10^1 . Thus, the following numbers of 0.0001278, 0.001278 and 0.01278 have the same number of floating points of four. These numbers can be written in the form of the floating point format as 0.1278×10^{-3} , 0.1278×10^{-2} and 0.1278×10^{-1} , respectively.

Numbers are stored using the binary system in computers. The binary system is different from the decimal or base-10 system. As an example, the value of 107 in the decimal system is determined from

$$1 \times 10^2 + 0 \times 10^1 + 7 \times 10^0 = 107$$

The same value is represented by 1101011 in the binary system, which is determined from

$$1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 107$$

The value of 1101011 in the binary system consists only the numbers 1 and 0 which are called bits. A total of 32 bits may be needed to represent a single value or a word. The value of 107 may be stored in the computer under the binary system as

$$000000000000000000000000000000001101011$$

In the above set of 32 bits, the first bit is used to indicate the positive or negative value of the number (0 = positive, 1 = negative). The following 31 bits are used to represent the number that can go up to $2,147,483,647$ (or $2^{31} - 1$). This means a typical 32 bit computer can store integer values ranging from $-2,147,483,647$ to $+2,147,483,647$. For a value that is beyond such range, it is stored in the floating point format. The 32 bits are divided into four parts. The first two parts represent the sign of the number and exponent, while the last two parts are used for storing the magnitudes of the exponent and mantissa. Thus, a typical word can store a floating point value in the range of -10^{38} to 10^{38} . Understanding how values are stored in the computers help users to be aware of the very small or large numbers during analyzing a problem.

As explained earlier, numerical error can arise from different sources. If an exact solution of the problem is known, the true error E_t can be determined. In the example of the space shuttle velocity, the true error is determined from

$$E_t = v_e - v_a \quad (1.22)$$

where v_e and v_a are the exact and approximate solutions, respectively. The true percentage error ε_t can also be computed as

$$\varepsilon_t = \frac{v_e - v_a}{v_e} \times 100\% \quad (1.23)$$

For example, the exact and approximate shuttle velocities at 30 seconds in Table 1.2 are 273 and 294 m/sec, respectively. Then, the true percentage error is

$$\varepsilon_t = \frac{273 - 294}{273} \times 100\% = -7.69\% \quad (1.24)$$

Because exact solutions are not available in practical problems, the approximate error is normally used to measure the solution error. For the example of the space shuttle velocity, the approximate error

may be defined by the difference between the two approximate solutions. In this case, the approximate percentage error ε_a is determined from

$$\varepsilon_a = \frac{v_{new} - v_{old}}{v_{new}} \times 100\% \quad (1.25)$$

where v_{new} and v_{old} are the two approximate velocities. For example, the two approximate velocities at 30 seconds obtained from using the time steps Δt of 30 and 10 seconds are 294 and 280 m/sec, respectively. Thus, the approximate percentage error is

$$\varepsilon_a = \frac{280 - 294}{280} \times 100\% = -5.00\% \quad (1.26)$$

1.6 Closure

The chapter presents an overview of the numerical methods for analyzing science and engineering problems. The chapter started from showing benefits of the methods for solving a variety of practical problems. Solutions obtained from the methods help analysts to understand behaviors that occur on the problems. Understanding the behaviors of the problems can lead to the improved designs. The chapter explained the key ingredients of the numerical methods that consist of the understanding of basic theories and the use of computer software and hardware. Several examples were presented to demonstrate advantages of the numerical methods for obtaining solutions as compared to the classical mathematics. In solving a simple problem, an approximate solution can be obtained easily by using a numerical method as compared to the finding of an exact solution from classical mathematics. For a more general problem, the exact solution is not available and the numerical method may be the only way to obtain a useful solution.

Different types of computers and computer languages widely used in the numerical methods for analyzing problems are explained. Popular languages normally employed in academic institutes and research organizations are Fortran, Pascal, C, Java and Python. Many computer software that include packages of numerical methods, such as MATLAB and Mathematica, are highlighted. The users, however, should understand the theories of the numerical methods behind these software prior to using them. For the cases where the software are not available and the users must develop computer programs by themselves, few computer commands must be understood. Simple computer programs using several languages are presented in this book to help readers in developing their own programs.

Exercises

1. Use Ex. 1.1 of the shuttle velocity determination to study the improved solution accuracy that will be obtained by reducing the time step. Compare the true errors by using the time steps of 10, 5 and 1 seconds. Tabulate the computed solutions and their errors between the times of 0 and 1,500 seconds with the increment of every 30 seconds.

2. Use the computer program in Ex. 1.3 to determine the computed solution accuracy of the sine and cosine functions by employing 3, 5 and 10 terms in the series. Determine the true and approximate errors by using 8 significant figures.
3. Develop a computer program to determine the exponential function which can be written in infinite series form

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

Then, use the program to determine the solutions of the function for $x = 0.1, 0.5, 1, 5$ and 50 with 8 significant figures. Explain the difficulties encountered and ways to improve the computed solutions.

4. Explain and develop a computer program to find roots of the equation,

$$ax^2 + bx + c = 0$$

where a, b and c are constants. The developed computer program should avoid problems that may occur from arbitrary values of a, b and c .

5. Develop computer programs to proof the following relations:

$$(a) 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$(b) 1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$(c) 1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$(d) 1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

In each case, use $n = 10, 50$ and 100 . Then, compute the true solution errors from the use of different values of n .

6. Develop computer programs to proof the following equalities:

$$(a) 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \dots = \ln 2$$

$$(b) 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots = \frac{\pi}{4}$$

$$(c) \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \dots = \frac{\pi^2}{6}$$

$$(d) \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \dots = \frac{\pi^2}{12}$$

$$(e) \quad \frac{1}{1^4} - \frac{1}{2^4} + \frac{1}{3^4} - \frac{1}{4^4} + \frac{1}{5^4} - \dots = \frac{7\pi^4}{720}$$

In each case, use the number of terms on the left-hand side of the equations as many as possible. Then, determine the true errors with 10 significant figures.

7. If $|x| < 1$, then the relation,

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \dots$$

is valid. Proof the validity of the relation by developing a computer program by using $x = 0.2$. Determine the true errors that occur from using 10, 50 and 100 terms on the right-hand side of the equation.

8. Develop a computer program to show that

$$\sum_{j=0}^n x^j = \frac{1-x^{n+1}}{1-x}$$

by using $x = 0.01, 0.1, 0.5, 0.9$ and 0.99 , respectively. Give comments on the accuracy of the computed solutions. Use the highest value of n that can be done on the computer.

9. Develop a computer program to proof that

$$\frac{1}{2} \ln\left(\frac{1+x}{1-x}\right) = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots$$

for $-1 < x < 1$. Use the number of terms on the right-hand side of equation as many as possible so that the computed solutions with 8 significant figures do not alter. Test the program by using $x = -0.5$ and 0.5 .

10. Develop a computer program for determining the value of π from

$$\sum_{n=1}^{\infty} \frac{1}{(2n-1)^2} = \frac{\pi^2}{8}$$

Compare the computed solution with the value of π in Eq. (1.21). Explain the difficulties encountered and suggest ways to improve accuracy of the computed solution.

11. Develop a computer program to show that

$$\sum_{n=1}^{\infty} \frac{1}{4n^2-1} = \frac{1}{2}$$

Then, determine the true percentage errors if $n = 10, 50$ and 100 . Show the computed solutions using 8 significant figures. Explain the difficulties encountered and suggest ways for improving the solutions.

12. Develop a computer program to show that

$$x^2 = \frac{c^2}{3} + \frac{4c^2}{\pi^2} \sum_{n=1}^{\infty} \frac{(-1)^n}{n^2} \cos \frac{n\pi x}{c}$$

where $0 \leq x \leq c$. If $c=4$, employ the developed program to verify the relation for $x=0.1, 1$ and 3 by using at least 8 significant figures in the computation.

13. Develop a computer program to determine the error function that is expressed in the form of infinite series as

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{n!(2n+1)}$$

Determine the function for $x=0.5, 1, 5$ and 10 by using 8 significant figures.

14. Develop computer programs to determine the Bessel functions of the first kind of order zero and one. These functions are expressed in the forms of infinite series as

$$J_0(x) = 1 - \frac{x^2}{2^2} + \frac{x^4}{2^2 \cdot 4^2} - \frac{x^6}{2^2 \cdot 4^2 \cdot 6^2} + \dots$$

$$J_1(x) = \frac{x}{2} - \frac{x^3}{2^2 \cdot 4} + \frac{x^5}{2^2 \cdot 4^2 \cdot 6} - \frac{x^7}{2^2 \cdot 4^2 \cdot 6^2 \cdot 8} + \dots$$

Determine the two functions for $x=1$ and 5 by using 5 significant figures. Compare the computed solutions with those tabulated in mathematical handbooks.

15. In an examination of a class that contains 40 students, the scores vary randomly between 0 and 100. Develop a computer program to reorder the scores of the students from 100 down to 0 with the corresponding student identities.

16. A wall with the thickness of π in x -direction has an initial temperature of zero. The wall is subjected to a uniform internal heat generation so that the transient temperature distribution with respect to time t is given by

$$T(x, t) = x + \frac{8}{\pi} \sum_{n=1}^{\infty} \frac{(-1)^n}{(2n-1)^2} \exp \left[-\frac{(2n-1)^2 t}{4} \right] \sin \frac{(2n-1)x}{2}$$

Develop a computer program to determine the transient temperature distribution. Plot the computed distributions at different times and show the distribution through the thickness of the wall when the time approaches infinity.

17. Buckling analysis of a vertical column due to its own weight leads to the need to determine the function

$$f(x) = 1 + \sum_{m=1}^{\infty} C_m x^{2m}$$

$$\text{where } m = 1 \quad ; \quad C_1 = -\frac{3}{8}$$

$$\text{and } m \geq 2 ; C_m = -\frac{3C_{m-1}}{4m(3m-1)}$$

Develop a computer program to determine the function for $0 \leq x \leq 2.0$ with the increment of x at every 0.2. Print the computed solutions by using 8 significant figures.

18. A solid sphere with radius of c has a uniform initial temperature of T_0 at time $t = 0$. If the outer surface temperature is changed abruptly to zero, the transient temperature distribution that varies with the radius r and time t is

$$T(r, t) = \frac{2T_0 c}{\pi} \sum_{n=1}^{\infty} (-1)^{n+1} \exp\left(-\frac{n^2 \pi^2 t}{c^2}\right) \frac{1}{nr} \sin \frac{n\pi r}{c}$$

Develop a computer program to compute and plot the radial temperature distributions at different times by using the value $c=5$ and $T_0=100$.

Chapter 2

Root of Equations

2.1 Introduction

In the process for solving many engineering and scientific problems, roots of equations are needed. If an equation is represented by a function $f(x)$, then the root x is such that it makes the value of the function to be zero. For example, the roots of the second-order polynomial,

$$f(x) = ax^2 + bx + c = 0 \quad (2.1)$$

where a, b and c are constants, are determined from the formula,

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (2.2)$$

Some polynomials are in higher-order form, such as

$$f(x) = 2x^4 - 7x^3 + 4x^2 + 7x - 6 = 0 \quad (2.3)$$

The factorization technique may be used to rewrite the polynomial into the form

$$(x-1)(x+1)(x-2)(2x-3) = 0 \quad (2.4)$$

so that the roots of the equation are obtained easily.

For general problems in engineering and scientific applications, the function $f(x)$ may not be in the form of polynomials as shown in Eq. (2.1) or (2.3). For examples, in the design for a proper cross-sectional area of a stop-sign pole, vibration of the pole due to the pressure load from the wind must be analyzed. In the analysis process, roots of the transcendental function in the form

$$f(x) = \cosh x \cos x + 1 = 0 \quad (2.5)$$

are required. Or in the analysis process for determining the shock wave angle generated from a 20 degree wedge, the root of the transcendental function

$$f(x) = 2 \cot x \left[\frac{9 \sin^2 x - 1}{9(1.4 + \cos 2x) + 2} \right] - \tan \frac{\pi}{9} = 0 \quad (2.6)$$

is needed. Roots of the transcendental equation,

$$f(x) = \sinh \frac{4}{9x} - \frac{5}{9x} = 0 \quad (2.7)$$

are required in the analysis process for determining tension in a cable that is suspended between two poles. In the buckling analysis of a vertical column due to its own weight, the root of the function in the form of an infinite series

$$f(x) = \sum_{n=1}^{\infty} \frac{3}{8} x^{2n} - 1 = 0 \quad (2.8)$$

is required. Or, in the example of the space shuttle in section 1.1, the velocity v of the vehicle during gliding into atmosphere is determined from

$$v = \frac{mg}{c} \left(1 - e^{-\frac{c}{m}t} \right) \quad (2.9)$$

where m is mass of the shuttle (90,000 kg), g is gravitational acceleration constant (9.81 m/sec^2), t is time in second, and c is the coefficient of drag force in kg/sec. At time $t = 500$ second and the shuttle velocity is 5 times speed of sound (about 1,650 m/sec) and if the coefficient of drag force c is needed, Eq. (2.9) leads to a transcendental equation in the form

$$1,650 = \frac{(90,000)(9.81)}{c} \left(1 - e^{-\frac{c}{90,000}(500)} \right) \quad (2.10)$$

The examples above are few problems that arise in engineering problems. These examples need to find values of x which are the roots of

$$f(x) = 0 \quad (2.11)$$

Popular methods to find such values are presented in this chapter. These methods are: (1) the graphical method, (2) the bisection method, (3) the false-position method, (4) the one-point iteration method, (5) the Newton-Raphson method, and (6) the secant method. All methods consist of simple computational procedures that can be understood easily. Moreover, these procedures can be used for developing computer programs directly.

2.2 Graphical Method

The simplest method for finding the root of function $f(x) = 0$ is the graphical method. The idea of this method is to determine values of the function at different x by plotting. The example below helps understanding the method procedure clearly.

Example 2.1 Find the root \bar{x} of the function,

$$f(x) = e^{-x/4}(2-x)-1 = 0 \quad (2.12)$$

by using the graphical method. Plot a graph to display behavior of the function.

Solution Values of the function $f(x)$ in Eq. (2.12) can be calculated and plotted as shown in Fig. 2.1

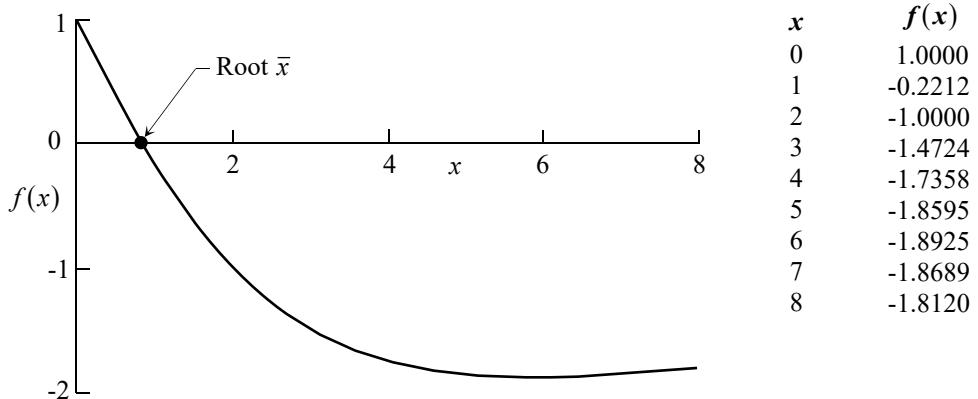


Figure 2.1 Distribution of $f(x)$ in Eq. (2.12) and its values at different x locations.

By considering Fig. 2.1, the function $f(x)$ is zero when it intersects the x -axis in the interval of $0.75 < x < 0.80$. After recalculating and plotting the function within this interval, the graph and its values are obtained as shown in Fig. 2.2.

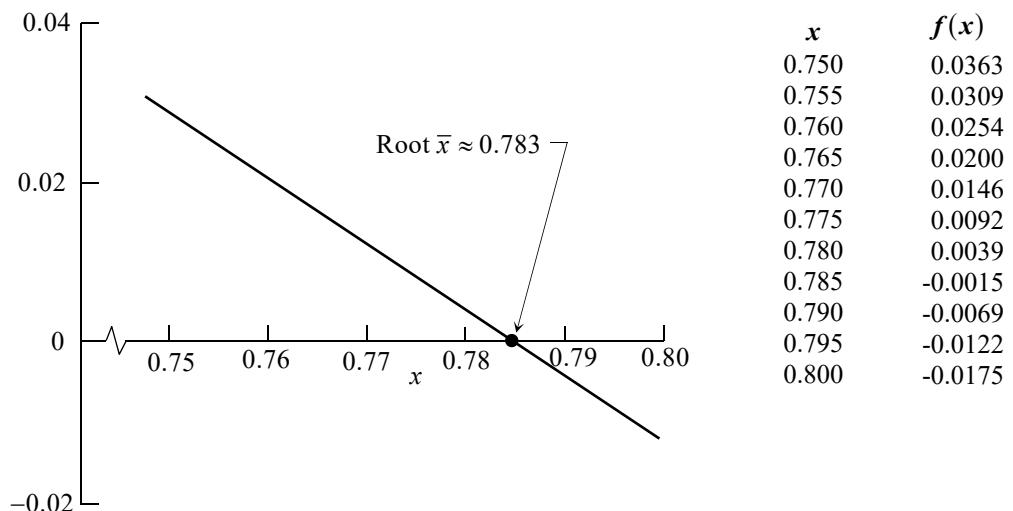


Figure 2.2 Distribution of $f(x)$ in Eq. (2.12) and its values at finer x locations.

From Fig. 2.2, a more accurate value of the root \bar{x} about 0.783 is obtained. It is an approximate value and may not be suitable if a solution with high accuracy is needed. However, the example shows that the graphical method is simple especially if a computer program for generating values of the function is available. Many commercial software today allow users to input a function, so that the software can generate values of the function and plot its variation directly on the monitor screen.

Even though the graphical method is easy but it is time consuming if a solution with high accuracy is needed. Other methods that can provide higher solution accuracy are presented in the following sections.

2.3 Bisection Method

The main idea of the bisection method is based on the fact that the values of the function have different signs when x are less and greater than the root \bar{x} as shown in Fig. 2.2. From example 2.1, the approximate value of the root \bar{x} is 0.783. When x is less than the root \bar{x} , the sign of the function is positive. The sign of the function becomes negative when x is greater than \bar{x} . In general, the sign of a function may change from a positive to negative value across the x -axis such as the graph in Fig. 2.2. The sign of a function may also change from a negative to positive quantity as shown in Fig. 2.3.

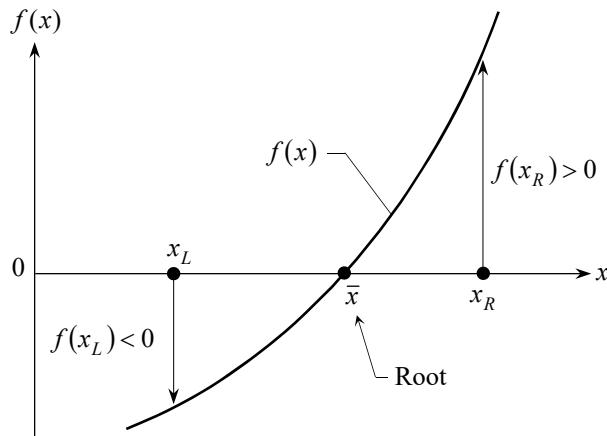


Figure 2.3 Bisection method for finding the root of $f(x)=0$.

Figure 2.3 shows that the function $f(x)$ changes from a negative value at $x = x_L$ (subscript L denotes Left side of \bar{x}) to a positive value at $x = x_R$ (subscript R denotes Right side of \bar{x}). The figure indicates that, as the sign of the values $f(x_L)$ and $f(x_R)$ are different, the root \bar{x} of the function must be between x_L and x_R .

The idea for finding root of the equation $f(x)=0$ by this method is to reduce the interval from x_L to x_R by a half and then properly select the sub-interval where the sign change occurs. This sub-interval, that contains the root \bar{x} , is used as a new interval for the next calculation. The computational procedure of the method are as follows.

Step 1 Find the mean value x_M from given x_L and x_R ,

$$x_M = \frac{x_L + x_R}{2} \quad (2.13)$$

Then determine the value $f(x_M)$ of the function at point x_M . Such value can be a positive (case A) or negative (case B) quantity as shown in Fig. 2.4.

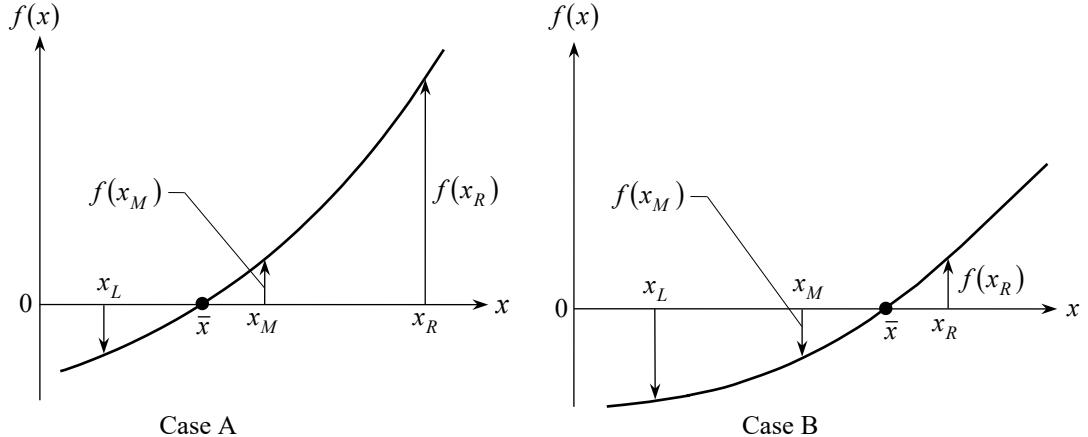


Figure 2.4 Value of $f(x_M)$ which can be a positive or negative quantity.

Step 2 Multiply $f(x_M)$ by $f(x_R)$,

If $f(x_M) \cdot f(x_R) > 0$ the result is case A where the root \bar{x} is in the interval $x_L < \bar{x} < x_M$.

If $f(x_M) \cdot f(x_R) < 0$ the result is case B, where the root \bar{x} is in the interval $x_M < \bar{x} < x_R$.

Step 3 Reassign the value of x_L or x_R to reduce the size of current interval.

If the result is case A, reassign value of x_R to x_M .

If the result is case B, reassign value of x_L to x_M .

Step 4 Check for convergence of the computed solution by using a criterion such as,

$$|f(x_M)| < \varepsilon \quad (2.14)$$

where ε is the acceptable error or tolerance. Another form of the convergence criterion is

$$\left| \frac{x_M^{new} - x_M^{old}}{x_M^{new}} \right| \times 100\% < \varepsilon_S \quad (2.15)$$

where ε_S is the stopping tolerance, for example 0.05%. If the computed result reaches the convergence criterion as specified in Eq. (2.14) or (2.15), the computation stops. If not, the computation is repeated by going back to step 1.

Example 2.2 Develop a computer program by using the four steps of the bisection method as explained earlier to find the root of Eq. (2.12). Use $x_L = 0$ and $x_R = 2$ with the stopping tolerance $\epsilon_S = 0.001\%$ in the form of Eq. (2.15).

A computer program to find the root of Eq. (2.12) by using the bisection method is shown in Fig. 2.5. The computed solutions at different iterations are presented in Table 2.1. The listing of computer program as shown in Fig. 2.5 can be modified to find roots of other equations. User can simply replace the function statement in the program by the function of a new problem.

```
% Program Bisect
% Program for computing root of nonlinear
% equation by using the bisection method.
% Define the following given values:
% xl = left value of the root x
% xr = right value of the root x
% es = stopping criterion tolerance (%)
% Define the function of the problem:
func = @(x)(exp(-x/4)*(2-x) - 1);
% Assign range and stopping tolerance:
xl = 0.; xr = 2.; es = 0.001;
% Check whether root is in given range:
fxl = func(xl); fxr = func(xr);
aa = fxl*fxr;
while aa >= 0.
    disp(' Root is not in the given range');
    break
end
fprintf('\n Iteration No. x \n');
for iter = 1:500
    xm = (xl+xr)/2.; fxm = func(xm);
    fxr = func(xr); aa = fxm*fxr;
    if aa > 0.
        % Case A: xl < root < xm
        xr = xm;
    else
        % Case B: xm < root < xr
        xl = xm;
    end
    % Check for the tolerance:
    xn = (xl+xr)/2.;
    fprintf(' %8d %22.6e\n', iter, xn);
    tol = abs((xn-xm)*100./xn);
    if tol < es
        fprintf('\n The root is %14.6e\n', xn)
        break
    end
    while tol > es
        fprintf(' Root cannot be reached for\n');
        fprintf(' the given range');
        break
    end
end
```

Figure 2.5 Computer program for finding the root of Eq. (2.12) by the bisection method.

Table 2.1 Solution convergence to the root of Eq. (2.12) by the bisection method.

Iteration number	Root x	Iteration number	Root x
1	0.500000	10	0.784180
2	0.750000	11	0.783691
3	0.875000	12	0.783447
4	0.812500	13	0.783569
5	0.781250	14	0.783630
6	0.796875	15	0.783600
7	0.789063	16	0.783585
8	0.785156	17	0.783592
9	0.783203		

In order to find a root of an equation by using the bisection method, user must know approximate position of root \bar{x} . The method needs a starting interval between x_L and x_R that contains root \bar{x} . The method keeps reducing the interval by a half until a converged solution is obtained. Because a starting interval between x_L and x_R is needed prior to the calculation, the method is sometimes called the bracketing method. The idea of the bisection method also leads to a more efficient method as presented in the next section.

2.4 False-position Method

The idea of the false-position method for finding the root of an equation is similar to that of the bisection method. Instead of bisecting the interval, the false-position method locates the root of equation by drawing a straight line to join the values of $f(x_L)$ and $f(x_R)$. The location x_1 that occurs from intersecting the straight line and the x -axis is the new estimated root of the equation as shown in Fig. 2.6. The figure gives the relation,

$$\begin{aligned} \tan \theta &= \tan \beta \\ \frac{f(x_R)}{x_R - x_1} &= \frac{f(x_L)}{x_L - x_1} \\ x_L f(x_R) - x_1 f(x_R) &= x_R f(x_L) - x_1 f(x_L) \end{aligned} \quad (2.16)$$

Thus, the location x_1 can be determined from

$$x_1 = \frac{x_L f(x_R) - x_R f(x_L)}{f(x_R) - f(x_L)} \quad (2.17)$$

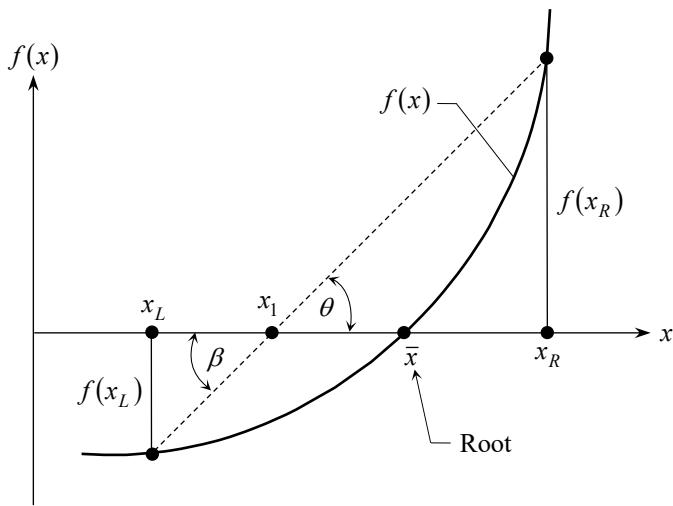


Figure 2.6 False-position method for finding the root of $f(x) = 0$.

The procedure of the false-position method starts from specifying the values of x_L and x_R . The location x_1 is then determined by using Eq. (2.17). Either the value of x_L or x_R is updated by a new appropriate value so that the current interval is reduced. Detailed steps of the false-position method are as follows.

Step 1 From the given locations x_L and x_R , determine the corresponding values of the function, $f(x_L)$ and $f(x_R)$. Then compute the location x_1 by using Eq. (2.17) and calculate the value of the

function $f(x_1)$ at this location. The computed value of the function can either be positive or negative as shown in Fig. 2.7.

Step 2 Multiply $f(x_1)$ by $f(x_R)$.

- | | |
|------------------------------|--|
| If $f(x_1) \cdot f(x_R) < 0$ | the result is case A where the root \bar{x} is in the interval $x_1 < \bar{x} < x_R$ |
| If $f(x_1) \cdot f(x_R) > 0$ | the result is case B where the root \bar{x} is in the interval $x_L < \bar{x} < x_1$ |

Step 3 Reassign the value of x_L or x_R according to the result from step 2.

If the result is in case A, reassign value of x_L to x_1 .

If the result is in case B, reassign value of x_R to x_1 .

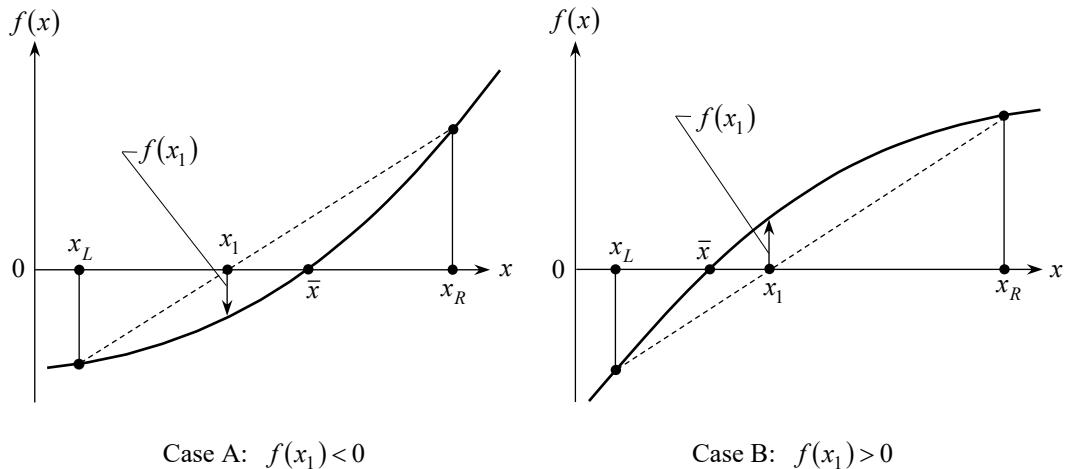


Figure 2.7 Value of $f(x_1)$ which can be positive or negative.

Step 4 Check for convergence of the computed solution by using a criterion as shown in Eq. (2.14) or (2.15). If the specified convergence criterion is met, stop the computation. If not, the computation is repeated by going back to step 1.

Example 2.3 Develop a computer program for finding the root of Eq. (2.12) by using the false-position method. Use the initial values $x_L = 0$ and $x_R = 2$ with the stopping tolerance ε_S in Eq. (2.15) as 0.001%.

The developed computer program is shown in Fig. 2.8. The computed solutions during the iteration process are shown in Table 2.2.

```

% Program FalPos
% Program for computing root of nonlinear
% equation by using the bisection method.
% Define the following given values:
%   xl = left value of the root x
%   xr = right value of the root x
%   es = stopping criterion tolerance (%)
% Define the function of the problem:
func = @(x) (exp(-x/4)*(2-x) - 1);
% Assign range and stopping tolerance:
xl = 0.; xr = 2.; es = 0.001;
% Check whether root is in given range:
fxl = func(xl); fxr = func(xr);
aa = fxl*fxr;
while aa >= 0.
    disp(' Root is not in the given range');
    break
end
x1old = xl;
fprintf('\n Iteration No.      x \n');
for iter = 1:500
    xm = (xl+xr)/2.;
    fxm = func(xm); fxr = func(xr);
    % Case A: xl < root < xr
    if xm < xr
        xl = xm;
    else
        xr = xm;
    end
    % Check for the tolerance:
    tol = abs((x1old-x1)*100./x1);
    if tol < es
        fprintf('\n The root is %14.6e\n', x1);
        break
    end
    x1old = x1;
end
while tol > es
    fprintf(' Root cannot be reached for\n');
    fprintf(' the given range');
    break
end

```

Figure 2.8 Computer program for finding the root of Eq. (2.12) by the false-position method.

Table 2.2 Solution convergence to the root of Eq. (2.12) by the false-position method.

Iteration number	Root x	Iteration number	Root x
1	1.000000	5	0.783741
2	0.818867	6	0.783619
3	0.789254	7	0.783600
4	0.784501	8	0.783597

By comparing convergence rates between the bisection and false-position methods as shown in Table 2.1 and 2.2, it is apparent that the false-position method converges faster than the bisection method. A schematic diagram for the solution convergence of the false-position method is shown in Fig. 2.8 (a).

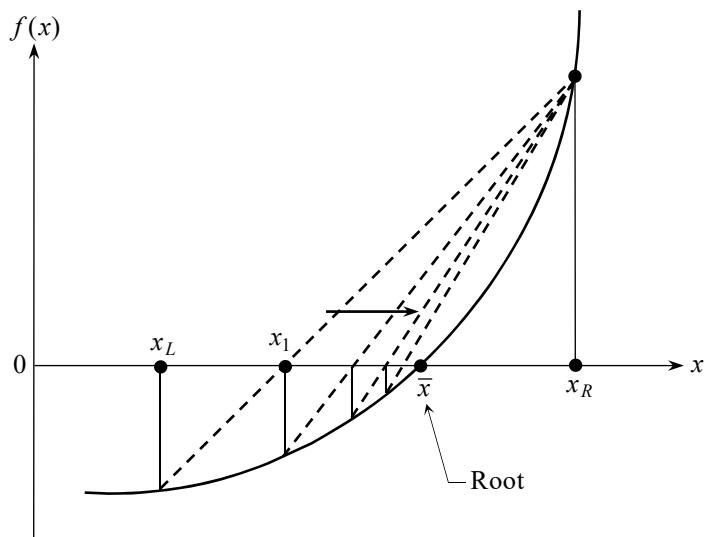


Figure 2.8(a) Schematic diagram for solution convergence of the false-position method.

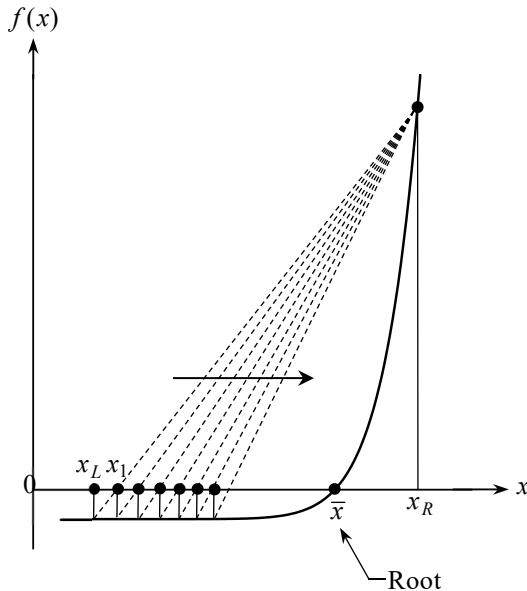


Figure 2.8(b) Slow convergence rate by the false-position method.

Figure 2.8 (a) shows that the specified positions of the initial locations x_L and x_R affect the convergence rate. If the initial location x_L in Fig. 2.8(a) moves to the left-hand side which is farther from the root \bar{x} , the number of iterations will increase. For some special functions such as that shown in Fig. 2.8(b), a large number of iterations is needed with low convergence rate. Both the bisection and false-position methods need an initial interval that contains the root of equation. Knowing a proper interval is the main advantage of the bracketing method because the iteration process always leads to a root of the equation. Before using these methods, user should have a priori knowledge of the proper interval that contains the root of equation.

For a practical problem, the location of the root \bar{x} is not known a priori. It will be very convenient for the computation if only one initial value of x is used for finding the root of equation. Such method is sometimes called the open method. Different types of the open method are presented in the following sections.

2.5 One-point Iteration Method

The one-point iteration method is one of the simplest open methods for finding the root of equation. The idea of the method starts by arranging the give function $f(x)=0$ such that the single variable x is placed on the left-hand side of the equation. For example, if the function $f(x)$ is given by

$$f(x) = \sinh 4x + 7x^2 - x + 3 = 0 \quad (2.18)$$

Equation (2.18) can be arranged to yield,

$$x = \sinh 4x + 7x^2 + 3 \quad (2.19)$$

Then, Eq. (2.19) is rewritten in an iterative form as,

$$x_{i+1} = \sinh 4x_i + 7x_i^2 + 3 \quad (2.20)$$

Equation (2.20) suggests that the new estimated value of root x_{i+1} is determined from the terms on the right-hand side of equation that use the old value of x_i .

If the given function $f(x)$ does not contain the single variable x that can be separated to the left-hand side of the equation, such as

$$f(x) = \cos x = 0 \quad (2.21)$$

In this case, the variable x can be added on both sides of the equation, so that

$$x = \cos x + x \quad (2.22)$$

Then, Eq. (2.22) is written in an iterative form as,

$$x_{i+1} = \cos x_i + x_i \quad (2.23)$$

Example 2.4 Use the one-point iteration method to find the root of Eq. (2.12) which is

$$f(x) = e^{-x/4}(2-x)-1 = 0$$

The equation above can be rewritten such that the single variable x is placed on the left-hand side of the equation as,

$$x = 2 - e^{x/4} \quad (2.24)$$

Equation (2.24) is then written in an iterative form as,

$$x_{i+1} = 2 - e^{x_i/4} \quad (2.25)$$

Equation (2.25) is used to develop a computer program as shown in Fig. 2.9. The computational procedure starts from an initial guess x as zero with the stopping tolerance of $\varepsilon_S = 0.001\%$. The computed solutions during the iteration process are shown in Table 2.3.

```
% Program OnePt
xold = 0.; es = 0.001;
fprintf('\n Iteration No.      x\n');
for i = 1:100
    xnew = 2. - exp(xold/4.);
    fprintf(' %8d    %14.6e\n',i,xnew);
    tol = abs((xnew-xold)*100./xnew);
    if tol < es
        break
    end
    xold = xnew;
end
```

Figure 2.9 Computer program for finding the root of Eq. (2.12) by the one-point iteration method.

Table 2.3 Solution convergence to the root of Eq. (2.12) by the one-point iteration method.

Iteration number	Root x	Iteration number	Root x
1	1.000000	7	0.783771
2	0.715975	8	0.783543
3	0.803987	9	0.783612
4	0.777379	10	0.783591
5	0.785485	11	0.783597
6	0.783021		

The concept for solution convergence of the one-point iteration method can be explained graphically. The function $f(x)=0$ in Example 2.4 can be separated into two equations; the first equation is $F(x)=x$ and the second equation is $G(x)=2-e^{x/4}$. The intersection point can be found by equating the two equations,

$$F(x) = G(x)$$

or

$$x = 2 - e^{x/4}$$

It can be seen that the equation obtained from the above procedure is identical to Eq. (2.24). Thus, it may be concluded that the concept of finding the root of equation by using the one-point iteration method is to find the intersection point between two equations, herein are $F(x) = x$ and $G(x) = 2 - e^{x/4}$ as shown in Fig. 2.10.

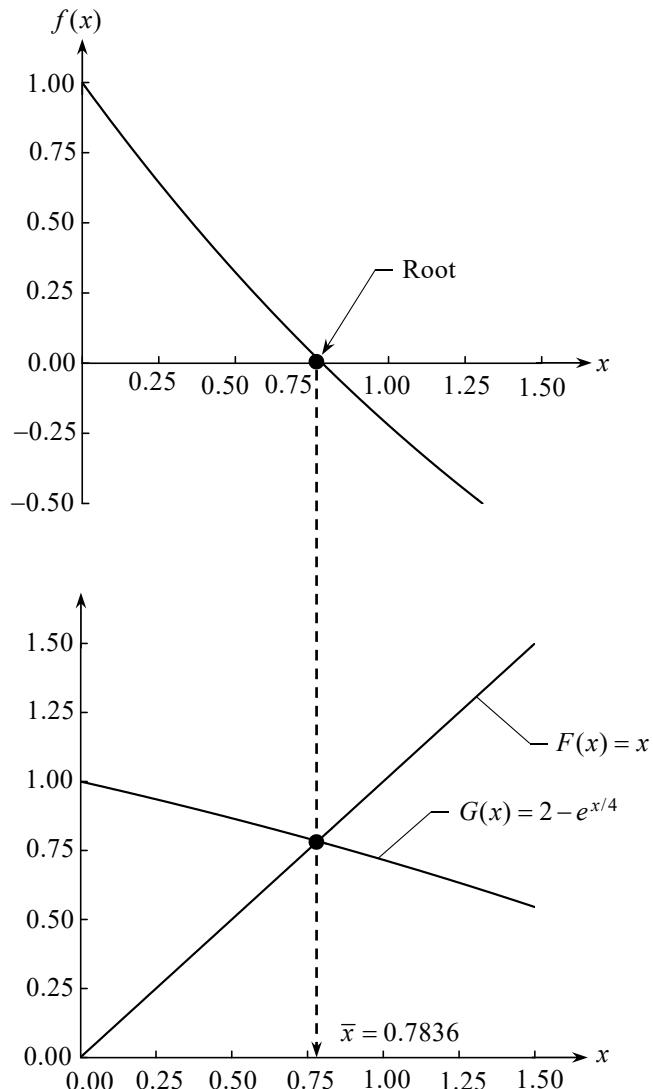


Figure 2.10 Root of equation by finding the intersection point of two functions.

Figure 2.11 shows the convergence behavior of the solution by the one-point iteration method. The initial guess value of x starts from zero (point ① in Fig. 2.11). Then, this value is used to calculate the expression on right-hand side of Eq. 2.25 which is point ① on the graph. Because the value on the left-hand side must be equal to the value on the right hand side, i.e., $F(x) = G(x)$, so the new estimated value of root is now moved to point ② which is placed on function $F(x) = x$. At this point, the value of the new estimated root is x_1 . After that, the point x_1 is used to calculate the expression on the right-hand side of Eq. (2.25) again that makes the solution converges to point ③ on the graph. When the functions $F(x)$ is set to be equal to $G(x)$, the new estimated value of root moves to point ④ with the value x_2 . The process is repeated until the solution is converged as a spiral shape to the intersection point between the function $F(x)$ and $G(x)$ which is the root of the equation $f(x)=0$. Another form for solution convergence of the one-point iteration method is the stair step pattern as shown by an example in Fig. 2.12. Figure 2.13 shows two examples that the one-point iteration method may lead to diverged solutions if the initial guess values are not provided properly.

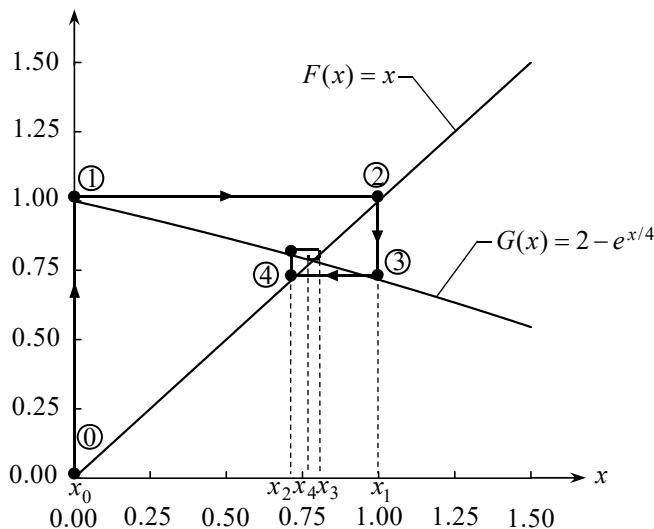


Figure 2.11 Convergence behavior of the solution by the one-point iteration method.

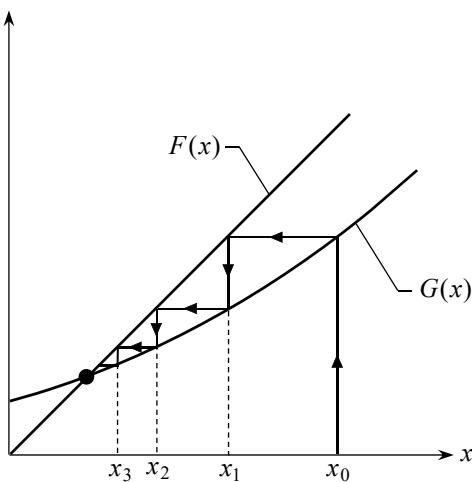


Figure 2.12 Convergence behavior of the solution by the one-point iteration method in the stair step pattern.

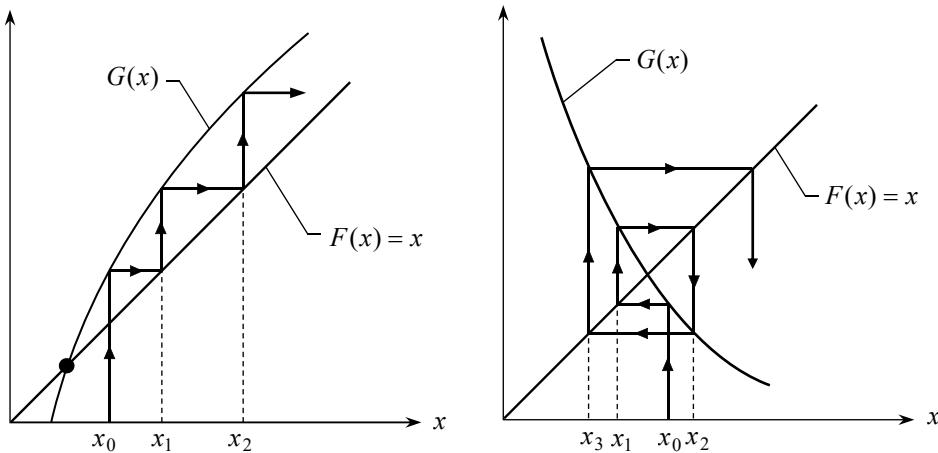


Figure 2.13 Schematic diagrams for solution divergence of the one-point iteration method.

2.6 Newton-Raphson Method

One of the most popular open methods that can provide rapid convergence for the root of equation is the Newton-Raphson method. Because the method is based on the use of Taylor series, thus, the concept of the series is explained herein first.

The Taylor series can be used to find the value of a function at point x from the values of the function and its derivatives at a nearby point x_0 . Even though the Taylor series contains an infinite number of terms but its physical meaning can be interpreted without difficulty. For example, if the Taylor series is approximated by using only the first term, i.e.,

$$f(x) \approx f(x_0) \quad (2.26)$$

it is called the zero-order approximation. Equation (2.26) implies that the value of the function at point x is approximated as its value at point x_0 . This approximation will be true if the considering function is constant. It is noted that Eq. (2.26) can also be used to approximate the value of the function at point x if the selected point x_0 is very closed to the point x .

If the first two terms of the Taylor series are used in the approximation,

$$f(x) \approx f(x_0) + (x - x_0) f'(x_0) \quad (2.27)$$

it is called the first-order approximation. Equation (2.27) is linear due to the first derivative term of $f'(x_0)$. Thus, the equation will provide exact solution if the considering function is linear.

Similarly, if the considering function varies quadratically, the Taylor series with the first three terms as shown in Eq. (2.28) will give the exact solution.

$$f(x) \approx f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) \quad (2.28)$$

In general, the functions that occur in engineering and scientific applications are complex and can not be represented by the polynomials with finite terms. To represent such complex function, the Taylor series with infinite terms must be used as,

$$f(x) = f(x_0) + (x-x_0)f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + \dots + \frac{(x-x_0)^n}{n!}f^{(n)}(x_0) + \dots \quad (2.29)$$

Representation of the Taylor series for a function is illustrated in the following example.

Example 2.5 Use the Taylor series to determine values of the function,

$$f(x) = \sin x \quad (2.30)$$

at point $x = \pi/6$ (or $x = 30^\circ$) by using the value of the function and its derivatives at point $x_0 = \pi/12$ (or $x = 15^\circ$). Determine the solutions by using the Taylor series with zero-order to sixth-order approximation.

It is noted that $\sin(\pi/6) = 0.5$ and

$$x - x_0 = \frac{\pi}{6} - \frac{\pi}{12} = \frac{\pi}{12}$$

If the zero-order approximation of the Taylor series with only one term is used as shown in Eq. (2.26), the approximate value of $\sin(\pi/6)$ is

$$f\left(\frac{\pi}{6}\right) \approx \sin\left(\frac{\pi}{12}\right) = 0.2588190$$

For the first-order approximation of the Taylor series with two terms as shown in Eq. (2.27), the approximate value of $\sin(\pi/6)$ is

$$f\left(\frac{\pi}{6}\right) \approx \sin\left(\frac{\pi}{12}\right) + \frac{\pi}{12} \cos\left(\frac{\pi}{12}\right) = 0.5116978$$

Similarly, for the third-order approximation of the Taylor series with three terms as shown in Eq. (2.28), the approximated value of $\sin(\pi/6)$ is

$$f\left(\frac{\pi}{6}\right) \approx \sin\left(\frac{\pi}{12}\right) + \frac{\pi}{12} \cos\left(\frac{\pi}{12}\right) - \frac{1}{2}\left(\frac{\pi}{12}\right)^2 \sin\left(\frac{\pi}{12}\right) = 0.5028282$$

The approximate values of $\sin(\pi/6)$ from the zero-order to sixth-order approximation are shown in Table 2.4. The table shows that the solution converges to the exact value which is 0.5 when the order of approximation is increased.

Table 2.4 Approximate values of $\sin(\pi/6)$ by Taylor series of order zero to six.

order n	$f^{(n)}(x)$	$f(\pi/6)$
0	$\sin x$	0.2588190
1	$\cos x$	0.5116978
2	$-\sin x$	0.5028282
3	$-\cos x$	0.4999396
4	$\sin x$	0.4999902
5	$\cos x$	0.5000001
6	$-\sin x$	0.5000000

The use of the Taylor series is the basis of the Newton-Raphson method for finding the root of equation $f(x)=0$. The idea of the method is to use the first two terms as shown in Eq. (2.27) for approximating the function $f(x)$ in the iteration process,

$$f(x) = f(x_0) + (x - x_0) f'(x_0) = 0 \quad (2.31)$$

or

$$(x - x_0) f'(x_0) = -f(x_0)$$

$$x - x_0 = -\frac{f(x_0)}{f'(x_0)} \quad (2.32)$$

The physical meaning of Eq. (2.32) is systematically explained by Fig. 2.14.

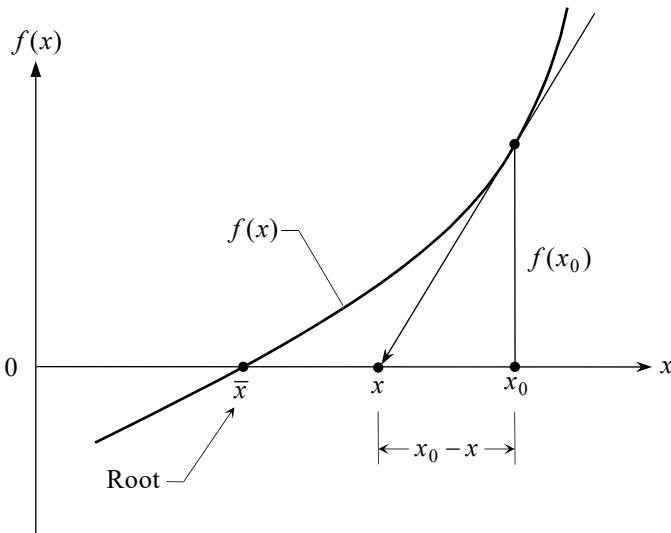


Figure 2.14 Finding the new approximate value x from the initial value x_0 by Newton-Raphson method.

The procedure of Newton-Raphson method starts from an initial value x_0 as shown in Fig. 2.14. Then, the values of function $f(x)$ and its first derivative $f'(x)$ at point x_0 are determined. These computed values are substituted into Eq. (2.32) to obtain the new value of x as shown in Fig. 2.14. The process is repeated until the new value of x converges to the root \bar{x} as shown in Fig. 2.15.

Figure 2.15 highlights a rapid convergence of the solution by using the Newton-Raphson method. The method is, thus, very popular among the methods for finding the root of an equation. However, the method may provide a solution that diverges from the real root. A diverged solution may be caused by the behavior of the function and the initial guess value x_0 , etc. Figure 2.16 (a-b) shows the Newton-Raphson method that can provide a converged or diverged solution depending on the initial guess value x_0 .

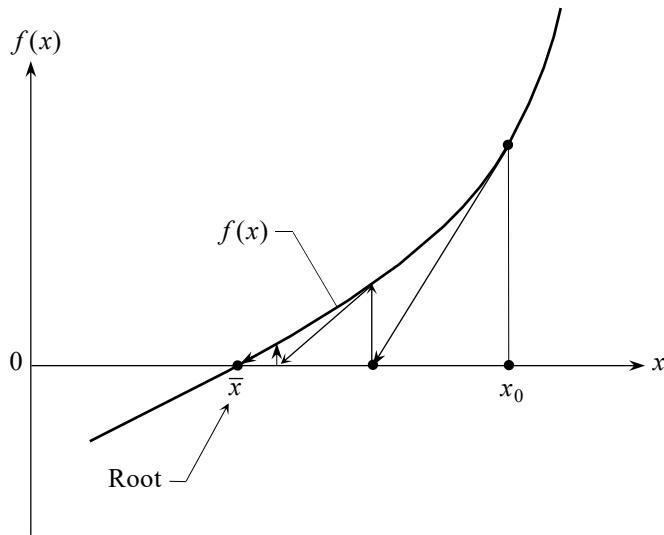


Figure 2.15 Convergence behavior of Newton-Raphson method.

From the above explanation, the method uses an old value to determine a new value of x through Eq. (2.32) which was approximated from the Taylor series. If Δx is the difference between the old and new values of x , Eq. (2.32) can be written as,

$$\Delta x = x - x_0 = -\frac{f(x_0)}{f'(x_0)} \quad (2.33)$$

The equation above can be used in developing a computer program with the following computational procedure.

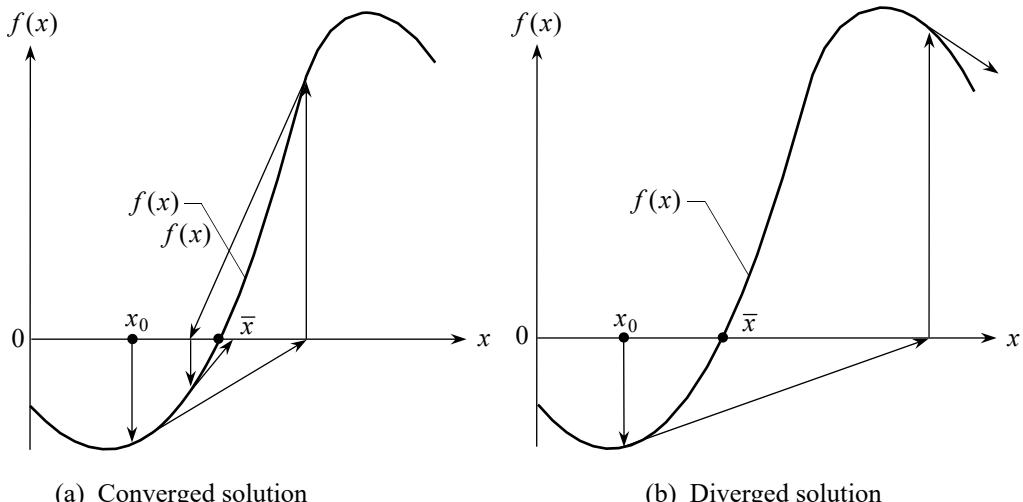


Figure 2.16 Convergence and divergence solution behaviors by the Newton-Raphson method.

Step 1 Determine the value of the function and its first derivative at the old point x . Then, determine the increment Δx from

$$\Delta x_{k+1} = -\frac{f(x_k)}{f'(x_k)} \quad (2.34)$$

where the subscripts k and $k + 1$ represent the iteration numbers k and $k + 1$, respectively.

Step 2 Determine the new value x from

$$x_{k+1} = x_k + \Delta x_{k+1} \quad (2.35)$$

Step 3 Check for the convergence of the solution by using one of the following convergence criteria,

$$(a) |\Delta x_{k+1}| < \varepsilon_1 \quad (2.36)$$

where ε_1 is the absolute error, or

$$(b) \left| \frac{\Delta x_{k+1}}{x_{k+1}} \right| < \varepsilon_2 \quad (2.37)$$

where ε_2 is the relative error, or

$$(c) \left| \frac{\Delta x_{k+1}}{x_{k+1}} \right| \times 100\% < \varepsilon_3 \quad (2.38)$$

where ε_3 is the percentage relative error. If the computed solution does not meet the specified convergence criterion, the process is repeated by going back to step 1.

Example 2.6 Develop a computer program to find the root of Eq. (2.12) by using the Newton-Raphson method. Use the initial guess $x_0 = 3$ and the percentage relative error in the form of Eq. (2.38) as $\varepsilon = 0.001\%$.

Before developing a computer program, the first derivative of the function is required. The given function in Eq. (2.12) is

$$f(x) = e^{-x/4}(2-x)-1$$

then

$$\begin{aligned} f'(x) &= e^{-x/4}(-1) + \left(-\frac{1}{4}\right)e^{-x/4}(2-x) - 0 \\ &= e^{-x/4}\left(-\frac{3}{2} + \frac{x}{4}\right) \end{aligned}$$

These equations are used in the development of a computer program as shown in Fig. 2.17. The computed solutions of x at different iterations are shown in Table 2.5.

```
% Program NewRap
% A program for computing root of a single
% equation using the Newton-Raphson method.
% Define the following given values:
% x0 = initial guess value of the root x
% es = stopping criterion tolerance (%)
x0 = 3.; es = 0.001;
% Define the function of the problem:
func = @(x) (exp(-x/4)*(2-x) - 1);
% and its derivative:
deriv = @(x) (-exp(-x/4)*(1.5-x/4));
fprintf('\n Iteration No.          x \n');
x = x0;
for iter = 1:500
    f = func(x); df = deriv(x); dx = -f/df;
    x = x + dx;
    fprintf(' %8d, %22.6e\n', iter, x);
    tol = abs(dx*100./x);
    if tol < es
        fprintf('\n The root is %14.6e\n', x)
        break
    end
end
while tol > es
    fprintf(' Root cannot be reached for\n');
    fprintf(' the given range');
    break
end
```

Figure 2.17 Computer program for finding the root of Eq. (2.12) by using the Newton-Raphson method.

Table 2.5 Solution convergence of the function in Eq. (2.12) by the Newton-Raphson method.

Iteration number	Root x	Iteration number	Root x
1	-1.156000	4	0.782542
2	0.189438	5	0.783596
3	0.714043	6	0.783596

2.7 Secant Method

The key idea of the secant method is the similar to the Newton-Raphson method except the first derivative of the function is obtained approximately. The method is useful when the given function is complex such that its exact derivative cannot be derived easily. The first derivative of the function is estimated from the values of the function at the two points x_0 and x_1 as shown in Fig. 2.18.

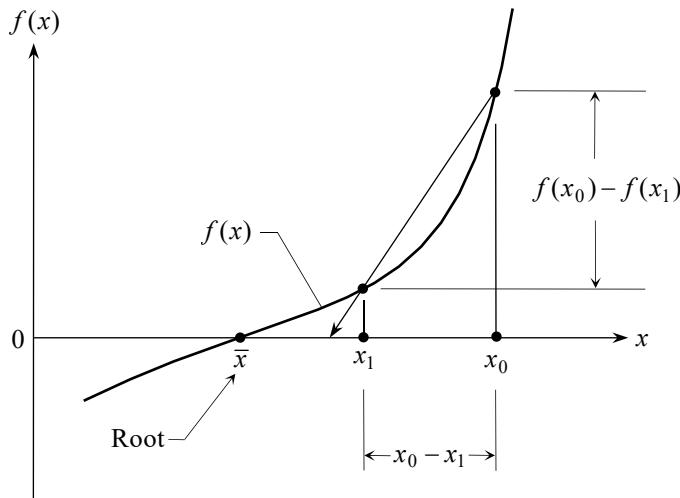


Figure 2.18 Finding the root of equation by the secant method.

$$f'(x_1) = \frac{f(x_0) - f(x_1)}{x_0 - x_1} \quad (2.39)$$

The approximate value in Eq. (2.39) is used to determine Δx through Eq. (2.33) of the Newton-Raphson method,

$$\Delta x = -\frac{f(x_1)}{f'(x_1)} = -\frac{f(x_1) \cdot (x_0 - x_1)}{f(x_0) - f(x_1)} \quad (2.40)$$

The computed Δx is then used for determining the new value of x . All other computational steps are identical to the Newton-Raphson method as shown in Eqs. (2.34) - (2.38).

Equation (2.39) and Fig. 2.18 show that the secant method requires two initial values of x to estimate the first derivative by using Eq. (2.39). The following example shows the results from the secant method for finding the root of Eq. (2.12).

Example 2.7 Develop a computer program for finding the root of Eq. (2.12) by using the secant method with the initial values $x_0 = 3$ and $x_1 = 2$. Use the convergence criterion $\varepsilon = 0.001\%$ as shown in Eq. (2.38).

The corresponding computer program is shown in Fig. 2.19 while the computed solutions during the iteration process are shown in Table 2.6.

```
% Program Secant
% A program for computing root of a single
% equation using the secant method.
% Define the following given values:
% x0 = first initial guess of root x
% x1 = second initial guess of root x
% es = stopping criterion tolerance (%)
x0 = 3.; x1 = 2.; es = 0.001;
% Define the function of the problem:
func = @(x) (exp(-x/4)*(2-x) - 1);
fprintf('\n Iteration No. \n');
for iter = 1:500
    f0 = func(x0); f1 = func(x1);
    df = (f0-f1)/(x0-x1);
    dx = -f1/df; x0 = x1; x1 = x1 + dx;
    fprintf(' %8d %22.6e\n', iter, x1);
    tol = abs(dx*100./x1);
    if tol < es
        fprintf('\n The root is %14.6e\n', x1)
        break
    end
end
while tol > es
    fprintf(' Root cannot be reached for\n');
    fprintf(' the given range');
    break
end
```

Figure 2.19 Computer program for finding the root of Eq. (2.12) by the secant method.

Table 2.6 Solution convergence to the root of Eq. (2.12) by the secant method.

Iteration number	Root x	Iteration number	Root x
1	-0.111700	4	0.781058
2	1.028830	5	0.783622
3	0.829093	6	0.783596

2.8 MATLAB Functions for Finding Root of Equation

Built-in functions in MATLAB for finding root of equation are explained in this section. The equation is input into MATLAB in the form of an anonymous function. The anonymous function has the following form,

```
f = @(variables) expression
```

where f is the name of the function
 expression is the expression of the function
 variables contain the independent variables in the function

For example, the function $f(x) = x^2 - 5$ can be created by using the command

```
>> f = @(x) x^2-5
```

MATLAB then responds as

```
f =
@(x) x^2-5
```

To find the value of the function, e.g., $x = 7$,

```
>> f(7)
ans =
44
```

If the function contains many variables, the same method can also be used. As an example,

```
>> g = @(x,y) 2+5*x+3*y
g =
@(x,y) 2+5*x+3*y
```

Root of function is determined by using the `fzero` command. The structure of the command is,

```
y = fzero(function,x0)
```

where y is the root value
 function is the function name
 x0 is the initial guess value

If the bracketing method is employed in the determination of the root, the two initial guess values x_0 and x_1 are specified by using the command,

```
y = fzero(function,[x0 x1])
```

Finding the root of an equation can be done easily by using the `fzero` command with an initial guess value. For example,

```
>> fzero(f,3)
ans =
2.2361
```

If a negative value of the roots is needed, a different value for the initial guess may be input,

```
>> fzero(f,-3)
ans =
-2.2361
```

Root of the equation may also be found by using two initial guess values. In this case, the command is

```
>> fzero(f, [-3 1])
ans =
-2.2361
```

If the root is not contained within the two given values, MATLAB will display the error message as follows,

```
>> fzero(f, [3 5])
??? Error using ==> fzero
The function values at the interval endpoints must differ in sign.
```

Example 2.8 Use the `fzero` command to find the root of Eq. (2.12) with the initial guess value of $x = 3$.

Before the root of equation is determined, the function in Eq. (2.12) must be input into MATLAB through the anonymous function. Then, the function `fzero` is used to find the root of the equation as follows,

```
>> f = @(x) exp(-x/4).* (2-x)-1;
>> y = fzero(f, 3)
y =
0.7836
```

It is noted that the `@()` command can be combined together with the `fzero` command to find the root of equation,

```
>> fzero(@(x) exp(-x/4).* (2-x)-1, 3)
ans =
0.7836
```

If the root of the equation is imaginary, the `fzero` command should not be used. Another popular command for finding the imaginary roots of a polynomial function is the `roots` command as,

```
y = roots(coef)
```

where `y` is the vector of the roots
`coef` is the vector of the polynomial coefficients

Example 2.9 Find the roots of the equation which is in the form of polynomials,

$$f(x) = x^4 - 9x^3 - 2x^2 + 120x - 130 = 0$$

The `roots` command is first used to create a vector that contains the coefficients of the polynomials as,

```
>> a = [1 -9 -2 120 -130];
```

The `roots` command is then employed to find the roots of the equation,

```
>> x = roots(a)
x =
7.3995
-3.6001
3.9721
1.2286
```

Example 2.10 Find the roots of the equation which is in the form of polynomials,

$$f(x) = x^4 - x^3 - 2.75x^2 + 5.25x - 2.5 = 0$$

The `roots` command can be used to find the real and imaginary roots of the equation as follows,

```
>> b = [1 -1 -2.75 5.25 -2.5];
>> x = roots(b)
x =
-2.0000
1.0000 + 0.5000i
1.0000 - 0.5000i
1.0000
```

2.9 Roots of System of Non-linear Equations

The topics presented earlier are for finding root of a single equation which is in the form of $f(x)=0$. In practice, analyses of engineering and scientific problems often lead to a system of non-linear equations. Fundamentals of the methods explained earlier can be used to find for the roots of such system of non-linear equations.

The system of non-linear equations consisting of n equations with n unknowns of x_1, x_2, \dots, x_n , can be written in a general form as,

$$\left. \begin{array}{l} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{array} \right\} n \text{ equations} \quad (2.41)$$

or in the matrix form,

$$[A] \{X\} = \{B\} \quad (2.42)$$

where $[A] = [A(x_i)]$ is the square matrix which contains the unknown variables x_i , $i = 1, 2, \dots, n$; $\{X\}$ is the column vector of the unknowns x_i , and $\{B\}$ is the vector that consists of known values. Examples of the matrices in Eq. (2.42) are shown in Example 2.11 and 2.12.

The methods for solving the system of non-linear Eq. (2.41) presented herein are: (1) the direct iteration method, and (2) the Newton-Raphson iteration method.

2.9.1 Direct iteration method

The main idea of direct iteration method for solving the system of non-linear equations is similar to the one-point iteration method for finding the root of a single equation as explained in section 2.5. The method starts from rewriting the functions in Eq. (2.41) in the form for performing iteration as,

$$\begin{aligned} x_1^{k+1} &= g_1(x_1^k, x_2^k, \dots, x_n^k) \\ x_2^{k+1} &= g_2(x_1^k, x_2^k, \dots, x_n^k) \\ &\vdots \\ x_n^{k+1} &= g_n(x_1^k, x_2^k, \dots, x_n^k) \end{aligned} \quad (2.43)$$

where k is the iteration number. The computational procedure of the method consists of the following steps.

Step 1 Provide the initial guess x_i^k , $i = 1, 2, \dots, n$

Step 2 Compute x_i^{k+1} , $i = 1, 2, \dots, n$ by using Eq. (2.43)

Step 3 Check for solution convergence of x_i by using the criterion such as,

$$\left| \frac{x_i^{k+1} - x_i^k}{x_i^{k+1}} \right| \times 100\% < \varepsilon \quad (2.44)$$

If the specified convergence criterion is not met, the process is repeated by going back to step 2.

Example 2.11 Use the direct iteration method for solving the system of non-linear equations in the matrix form below,

$$\begin{bmatrix} -1 & 1 \\ 1 & x_2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 5 \end{Bmatrix} \quad (2.45)$$

The two equations in Eq. (2.45) are first expressed as,

$$-x_1 + x_2 = 1 \quad (2.46a)$$

$$x_1 + x_2^2 = 5 \quad (2.46b)$$

Equations (2.46a-b) are then written in the iteration form,

$$x_1^{k+1} = x_2^k - 1 \quad (2.47a)$$

$$x_2^{k+1} = \sqrt{5 - x_1^k} \quad (2.47b)$$

By providing the initial guess of $x_1 = x_2 = 0$, the new values of x_1 and x_2 are determined from Eqs. (2.47a-b). The computed solutions of x_1 and x_2 at each iteration are shown in Table 2.7.

Table 2.7 Convergence of solutions to the roots of Eq. (2.45) by the direct iteration method.

Iteration number	x_1	x_2
0	0.000	0.000
1	-1.000	2.236
2	1.236	2.449
3	1.449	1.940
4	0.940	1.884
.	.	.
.	.	.
.	.	.
.	.	.
12	1.000	2.000

2.9.2 Newton-Raphson Iteration Method

The basic idea of the Newton-Raphson method for solving a set of non-linear equations in the form of Eq. (2.41) is similar to that for finding the root of a single equation as explained in section 2.6. The only difference is the use of the Taylor series for n variables. For a typical equation i , the Taylor series for n variables is given by

$$\begin{aligned} f_i(x_1 + \Delta x_1, x_2 + \Delta x_2, \dots, x_n + \Delta x_n) \\ = f_i(x_1, x_2, \dots, x_n) + \sum_{j=1}^n \frac{\partial f_i}{\partial x_j}(x_1, x_2, \dots, x_n) \Delta x_j + \dots \end{aligned} \quad (2.48)$$

where x_i , $i = 1, 2, \dots, n$ are the values used for determining the new values of $x_i + \Delta x_i$. If only the first two terms on the right-hand side of Eq. (2.48) are used in the approximation, the resulting equation will be in the similar form as shown in Eq. (2.31), i.e.,

$$0 = f_i(x_1, x_2, \dots, x_n) + \sum_{j=1}^n \frac{\partial f_i}{\partial x_j}(x_1, x_2, \dots, x_n) \Delta x_j \quad (2.49)$$

or,

$$\sum_{j=1}^n \frac{\partial f_i}{\partial x_j} \Delta x_j = -f_i \quad (2.50)$$

For example, if a system of non-linear equations consists of only 3 equations, then $n = 3$ and $i, j = 1, 2, 3$. Equation (2.50) can be written explicitly as,

$$\left[\begin{array}{ccc} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{array} \right] \left[\begin{array}{c} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{array} \right] = - \left[\begin{array}{c} f_1 \\ f_2 \\ f_3 \end{array} \right] \quad (2.51)$$

For a set of n non-linear equations, the Newton-Raphson iteration method leads to,

$$[J] \begin{matrix} \{\Delta x\} \\ (n \times n) \end{matrix} = -\{f\} \quad (2.52a)$$

where $[J]$ is called the Jacobian matrix for which its coefficients are determined from

$$J_{ij} = \frac{\partial f_i}{\partial x_j} \quad (2.52b)$$

The vector $\{\Delta x\}$ contains increments of the solutions. The vector $\{f\}$ on the right-hand side of the equation consists of the function f_i , $i = 1, 2, \dots, n$ which are evaluated at x_i . This vector $\{f\}$ is sometimes called the residual vector because it becomes zero as all x_i converge to the correct solutions.

From the above explanation, the computational procedures of the Newton-Raphson iteration method for finding the roots of non-linear equations are as follows.

Step 1 Solve the system of equations,

$$[J]^k \{\Delta x\}^{k+1} = -\{f\}^k \quad (2.53)$$

where the superscript k denotes the iteration number.

Step 2 Compute the new solution values from

$$\{x\}^{k+1} = \{x\}^k + \{\Delta x\}^{k+1} \quad (2.54)$$

Step 3 Check for solution convergence by using the criterion as shown in Eq. (2.36) – (2.38). If the computed solutions are not converged within the specified criterion, the process is repeated by going back to step 1.

Example 2.12 Use the Newton-Raphson iteration method to solve the system of non-linear equations given in matrix form,

$$\begin{bmatrix} 1 & 2 & 1 & 4 \\ x_1 & 2x_1 & 0 & x_4^2 \\ x_1^2 & 0 & x_3 & 1 \\ 0 & 3 & 0 & x_3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 20.700 \\ 15.880 \\ 21.218 \\ 21.100 \end{Bmatrix} \quad (2.55)$$

Solution From Eq. (2.55), the four functions are,

$$\begin{aligned} f_1 &= x_1 + 2x_2 + x_3 + 4x_4 - 20.700 \\ f_2 &= x_1^2 + 2x_1x_2 + x_4^3 - 15.880 \\ f_3 &= x_1^3 + x_3^2 + x_4 - 21.218 \\ f_4 &= 3x_2 + x_3x_4 - 21.100 \end{aligned} \quad (2.56)$$

It is noted that if the correct solutions of x_1 , x_2 , x_3 and x_4 are substituted into the Eq.(2.56), the four functions of f_i , $i = 1, 2, 3, 4$ are all zero.

The size of the Jacobian matrix for this problem is (4×4) and its coefficients are determined from Eq. (2.52b) as,

$$[J] = \begin{bmatrix} 1 & 2 & 1 & 4 \\ 2x_1 + 2x_2 & 2x_1 & 0 & 3x_4^2 \\ 3x_1^2 & 0 & 2x_3 & 1 \\ 0 & 3 & x_4 & x_3 \end{bmatrix} \quad (2.57)$$

If the initial guess values are $x_1 = x_2 = x_3 = x_4 = 1$, then Eq. (2.53) becomes,

$$\begin{bmatrix} 1 & 2 & 1 & 4 \\ 4 & 2 & 0 & 3 \\ 3 & 0 & 2 & 1 \\ 0 & 3 & 1 & 1 \end{bmatrix} \begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \end{Bmatrix} = -\begin{Bmatrix} -12.700 \\ -11.880 \\ -18.218 \\ -17.100 \end{Bmatrix} \quad (2.58)$$

The procedure for solving a system of Eq. (2.58) will be explained in details in chapter 3. The solutions for the system of equations above are,

$$\begin{Bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \\ \Delta x_4 \end{Bmatrix} = \begin{Bmatrix} 1.75037 \\ 3.67630 \\ 6.89579 \\ -0.82469 \end{Bmatrix} \quad (2.59)$$

Thus, the new values of the roots are computed by using Eq. (2.54),

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 1. \\ 1. \\ 1. \\ 1. \end{Bmatrix} + \begin{Bmatrix} 1.75037 \\ 3.67630 \\ 6.89579 \\ -0.82469 \end{Bmatrix} = \begin{Bmatrix} 2.75037 \\ 4.67630 \\ 7.89579 \\ 0.17531 \end{Bmatrix} \quad (2.60)$$

Convergence of the solutions is then checked by using the specified criterion. If the convergence criterion is not met, the procedure is repeated by determining the Jacobian matrix in Eq. (2.57) with the new computed values from Eq. (2.60). Table 2.8 shows the computed solutions with 6 significant figures that are obtained different iterations. The final solutions can be verified by substituting them back into the functions in Eq. (2.56) so that all functions must be zero.

Table 2.8 Converged solutions to the roots of the non-linear equations in Example 2.12.

Iteration number	x_1	x_2	x_3	x_4
0	1.00000	1.00000	1.00000	1.00000
1	2.75037	4.67630	7.89579	0.17531
2	1.34485	5.29712	5.94935	0.70289
3	1.47750	3.84372	4.34185	1.79830
.
.
.
.
8	1.20000	5.60000	4.30000	1.00000

2.10 Closure

In this chapter, the methods for finding roots of a single equation and a set of non-linear equations were presented. By understanding the procedure of these methods together with ability to develop the corresponding computer programs, the roots of the equation can be obtained easily. Finding the root of a single equation is an important topic that occurs in the analysis of many engineering problems. Understanding the methods can help analysts to determine and verify the computed solutions.

Some of the methods used for finding solution of a single equation were extended to solve for solutions of a set of non-linear equations. The set of non-linear equations often occurs in the analysis of practical applications. Some popular methods, such as the Newton-Raphson iteration method, have been used and built inside the commercial software today. Software users need to understand the procedures for solving these non-linear equations to assure the accuracy of the computed solutions.

Exercises

1. In the vibration analysis of a stop sign pole, the root of the following transcendental function is required,

$$\cosh x \cos x + 1 = 0$$

Find the first three positive roots by using: (a) the graphical method, (b) the bisection method, (c) the false-position method and (d) the Newton-Raphson method. Use appropriate initial guess values in order to obtain the solution with 6 significant figures.

2. Employ the bisection method to determine the value of $\sqrt[4]{13}$ by solving the equation,

$$x^4 - 13 = 0$$

Use the initial left and right values of 1.5 and 2.0. Perform iteration until the solution with six significant figures does not alter.

3. Employ the false-position method to determine the value of $1/43$ by solving the following equation

$$\frac{1}{x} - 43 = 0$$

Use the initial left and right values of 0.02 and 0.03. Determine the solution by using the convergence criterion with the tolerance of 0.000001%.

4. Apply the Taylor series to determine the value of the function

$$f(x) = \ln x$$

at $x = 4$ by using the value of the function and its derivatives at $x = 2$. Determine values of the function from the Taylor series of order 0 to 6 and compute the errors for each case.

5. Apply the Taylor series and develop a computer program to determine the value of the function

$$f(x) = e^{-x}$$

at $x = 2$ by using the value of function and its derivatives at $x = 1$. Determine values of the function from the Taylor series of order 0 to 6 and compute the errors for each case.

6. In the determination for the oblique shock angle β generated from a flow of Mach M over an inclined plan that makes angle θ with the horizontal axis, the root of the equation,

$$\tan \theta - 2 \cot \beta \left[\frac{M^2 \sin^2 \beta - 1}{M^2 (\gamma + \cos 2\beta) + 2} \right] = 0$$

is needed. In the above equation, γ is the specific heat ratio for air which is 1.4. The flow is at Mach $M = 3$ and $\theta = \pi/9$. Determine the angle β of the oblique shock by using: (a) the bisection method with the initial left and right values of 0 and $\pi/4$, and (b) the secant method with the initial left and right values of $\pi/6$ and $\pi/4$. The computed solution of the angle β must have accuracy up to 6 significant figures.

7. Find the roots of the following equations by using the Newton-Raphson method. The numbers in parentheses are the initial guess values for each case. The computed solutions must have their accuracy up to 4 significant figures.

$$(a) \quad x^2 - 2x - 3 = 0 \quad (0.5)$$

$$(b) \quad xe^{-x^2} - \cos x = 0 \quad (2.0)$$

$$(c) \quad 10 \ln x - x = 0 \quad (1.0)$$

$$(d) \quad e^x - (1 + x + x^2/2) = 0 \quad (1.5)$$

$$(e) \quad x^3 - 100 = 0 \quad (2.0)$$

$$(f) \quad e^x - \sin(\pi x/3) = 0 \quad (-2.8)$$

8. Find the value of $\sqrt{7}$ with its accuracy up to 8 significant figures by using the Newton-Raphson method. It is noted that the solution can be obtained from the equation,

$$x^2 - 7 = 0$$

Use the initial guess of $x = 2$.

9. Use the Newton-Raphson method to determine the eigenvalue that occurs in the buckling analysis of a vertical bar due to its own weight. The eigenvalue β is the root of the equation,

$$1 + \sum_{m=1}^{\infty} C_m \beta^{2m} = 0$$

where $m = 1$ $C_1 = -\frac{3}{8}$

and $m \geq 2$

$$C_m = -\frac{3C_{m-1}}{4m(3m-1)}$$

Use the initial guess of zero and the computed solution must have their accuracy up to 6 significant figures.

10. In the vibration analysis of a beam that has a mass attached at the middle of it, the root of the equation,

$$2 - x(\tan x - \tanh x) = 0$$

is needed. Use: (a) the bisection method, (b) the Newton-Raphson method and (c) function `fzero` of MATLAB for finding the root. The computed solutions must have their accuracy up to 6 significant figures. Hint: the root of this equation is between 0.5 and 1.5.

11. A cable with the length $\ell = 180$ m and weight $w = 36$ N/m hangs between two poles. The distance between the two poles is $L = 165$ m. The tension H in the cable is determined from

$$\sinh \frac{wL}{2H} = \frac{w\ell}{2H}$$

Use: (a) the Newton-Raphson method, (b) the secant method and (c) function `fzero` of MATLAB to find the tension H in the cable. The computed solutions must have their accuracy up to 6 significant figures.

12. Find the drag coefficient of the space shuttle as shown in Eq. (2.10) by using: (a) the bisection method, (b) the false-position method and (c) the Newton-Raphson method. The computed solutions must have their accuracy up to 6 significant figures.

13. Show that the equation,

$$x^4 - 9x^3 - 2x^2 + 120x - 130 = 0$$

has four roots. Then, use any method explained in this chapter to find those roots with their accuracy up to 8 significant figures.

14. Derive related equations and explain computational procedures for determining the value of $\sqrt[3]{C}$ by using the Newton-Raphson method. The value C is a positive real number. Then develop a computer program to determine the value C by using the stopping criterion of $\varepsilon = 0.000001\%$ as shown in Eq. (2.38).

15. Use the Newton-Raphson method to find the root of the equation,

$$\cos x = x e^x$$

with the initial guess $x = 1$. Perform iteration until the solution with six significant figures does not alter.

16. Show that the n^{th} root of any constant C can be obtained by solving the equation

$$x^n - C = 0$$

Also show that the equation for finding the root of the above equation by the Newton-Raphson method is in form

$$x_{k+1} = \frac{1}{n} \left[(n-1)x_k + \frac{C}{x_k^{n-1}} \right]$$

where the subscript k is the iteration number.

17. Use the computer program of the secant method to find the root of the equation in Example 2.7 with initial guess values $x_0 = 3$ and $x_1 = 1$. Discuss the convergence behavior as compared to the solution in Example 2.7. Plot graph to compare the convergence rates of the two solutions.
18. Energy loss due to friction of flow in a pipe is required for selecting an appropriate size of the water pump. If the pipe has diameter D and length L , the energy loss is determined from,

$$h_l = f \frac{L}{D} \frac{\bar{V}^2}{2}$$

where h_l is energy loss, \bar{V} is the average flow velocity and f is the friction factor. If the inside surface of pipe is not smooth, the friction factor is determined from,

$$\frac{1}{f^{0.5}} = -2 \log \left(\frac{\varepsilon}{3.7} + \frac{2.51}{Re f^{0.5}} \right)$$

where ε is the roughness value of the surface and Re is the Reynold number. Use any built-in function of MATLAB to find the value of friction factor when $\varepsilon = 0.01$ and $Re = 100,000$. Compare the computed solution with the solution obtained from any method explained in this chapter.

19. A cable was hung between the two points at the same level. The distance between the two points is 250 m as shown in Fig. P2.19.

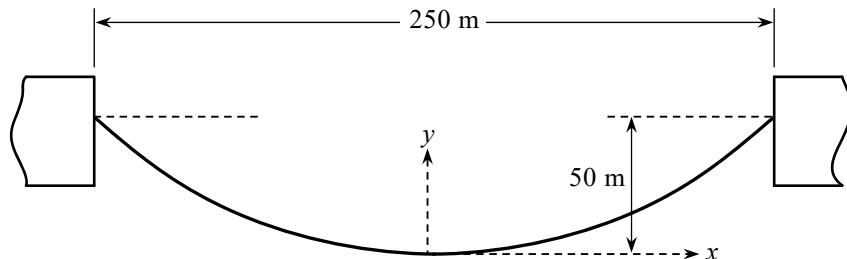


Figure P2.19.

The cable deflects by its own weight in a parabolic shape and the maximum deflection occurs at middle of the cable. The deflection curve for the cable is in the form,

$$y = \frac{T_0}{w} \left(\cosh \frac{wx}{T_0} - 1 \right)$$

where y is the deflection distance, w is the cable weight per unit length, T_0 is the cable tension at the position of maximum deflection, and x is the coordinate in the horizontal direction. If $w = 120 \text{ N/m}$ at the position $x = 125 \text{ m}$ and the deflection $y = 50 \text{ m}$, use MATLAB to find the value of the tension T_0 . Compare the computed solution with the solution obtained from the one-point iteration method.

20. A beam as shown in Fig. P2.20 is subjected to the load which increases along the beam length. The equation for the beam deflection curve is,

$$y = -\frac{wx}{360EI L} (3x^4 - 10L^2x^2 + 7L^4)$$

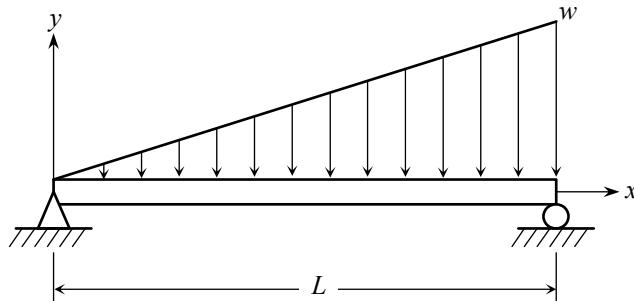


Figure P2.20.

where y is the deflection of beam, x is the distance along the beam axis, w is the load along beam, E is the Young's modulus, I is the moment of inertia of the beam cross-section and L is the beam length. Employ MATLAB to find the location of the maximum deflection that occurs along the beam and its value. Use the following data of $L = 450 \text{ cm}$, $E = 50,000 \text{ kN/cm}^2$, $I = 30,000 \text{ cm}^4$ and $w = 1.75 \text{ kN/cm}$ in the analysis. It is noted that the maximum deflection occurs at the location of $dy/dx = 0$.

21. Use the `roots` command in MATLAB to find all roots of the polynomial equation,

$$x^5 - 5x^4 - 35x^3 + 125x^2 + 194x - 280 = 0$$

Compare the solutions with the results obtained from any method in this chapter. Then, explain advantages and disadvantages of each method.

22. Use the `roots` command in MATLAB to find all roots of the polynomial equation,

$$x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25 = 0$$

Compare the solutions with the results obtained from any method in this chapter.

23. Develop a computer program by using the direct iteration method for solving the system of non-linear equations,

$$x^3 - 6y = 2$$

$$5x + 2y^2 = 12$$

Then, compare the computed solutions with the exact solutions.

24. Use the Newton-Raphson method to find the roots for the system of non-linear equations with 3 significant figures,

$$x^2y - xy^2 + 6 = 0$$

$$y^2 + 4x^2 - 3xy - 7 = 0$$

25. Use the Newton-Raphson method to find the roots for the system of non-linear equations

$$xy = 1 \quad \text{and} \quad y = \cos h x$$

Show the detailed Jacobian matrix and use the initial values $x = y = 1$ to compute the solutions. Perform 5 iterations and compare the computed results with the exact solutions.

26. Use the Newton-Raphson method to find the roots for the system of non-linear equations,

$$3x^2 + 4y - 5yz = -19$$

$$-xz + 2y^2 + z^2 = 14$$

$$xy^2 - 3yz + x^2z = -11$$

Employ the initial values $x = y = z = 2$ and show detailed computation with 4 significant figures for the first iteration. Perform 5 iterations and present the computed solutions in a table.

27. Develop a computer program by using the Newton-Raphson method for solving the system of non-linear equations,

$$5x_1x_3 - 2x_1x_2 + 4x_3^2 - x_2x_4 = 9.75$$

$$6x_1 + 3x_2 + x_3 - x_4 = 5.50$$

$$2x_1^2 + x_2x_3 - 5x_3 + x_1x_4^2 = -3.50$$

$$-3x_1x_4 - 2x_2^2 + 6x_3x_4 + x_3x_4 = 16.00$$

Use the initial guess values $x_1 = x_2 = x_3 = x_4 = 1$ and perform the computation until the solutions converge to 6 significant figures of accuracy.

Chapter

3

System of Linear Equations

3.1 Introduction

In chapter 2, several techniques for finding root of a single equation often occurs during the analysis of engineering problems were studied. For practical problems, many numerical techniques such as the finite difference and finite element techniques are used for obtaining solutions that describe their physical behaviors. In the analysis process, these methods lead to a set of simultaneous algebraic equations for which their roots representing the solutions of the problems are required. For examples, the finite difference technique was used to determine the flow phenomena surrounding a fighter jet in Fig. 1.1, while the finite element technique was employed to predict the structure deformation of passenger car during its collision in Fig. 1.2. During the analyses of these two problems, both techniques generate the large sets of algebraic simultaneous equations. The unknowns of the problems which are the roots for the sets of the algebraic equations must be solved. Understanding the methods for solving such system of equations together with the ability for developing computer programs are thus very important in the process for solving practical engineering problems.

In this chapter, various methods for solving the system of equations will be explained. Advantages and disadvantages of each method will be described. To illustrate the computational procedure for each method clearly, a small size of a system of equations will be used as an example. However, the computational procedures can be applied to solve a large size of the system of equations.

To demonstrate that a set of equations is arisen during the analysis of a typical problem, the following example is studied.

Example 3.1 A square metal plate, with the thickness t and thermal conductivity k , is subjected to a surface heating q as shown in Fig. 3.1. The origin of the x - y coordinates is at the plate center as shown in the figure. Zero temperature is specified as the boundary conditions along the four edges of the plate.

With the specified surface heating and boundary conditions, high temperature occurs at the plate center with the temperature distribution $T(x, y)$ as shown in the figure.

The governing equation that represents the conservation of energy for determining the temperature distribution over the plate is

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = -\frac{q}{kt} \quad (3.1)$$

If the size of the plate is 4×4 unit and $q/kt = 400$, the exact temperature distribution is

$$T(x, y) = 200(4 - x^2) - \frac{25,600}{\pi^3} \sum_{n=0}^{\infty} \frac{(-1)^n \cos((2n+1)\pi x/4) \cosh((2n+1)\pi y/4)}{(2n+1)^3 \cosh((2n+1)\pi/2)} \quad (3.2)$$

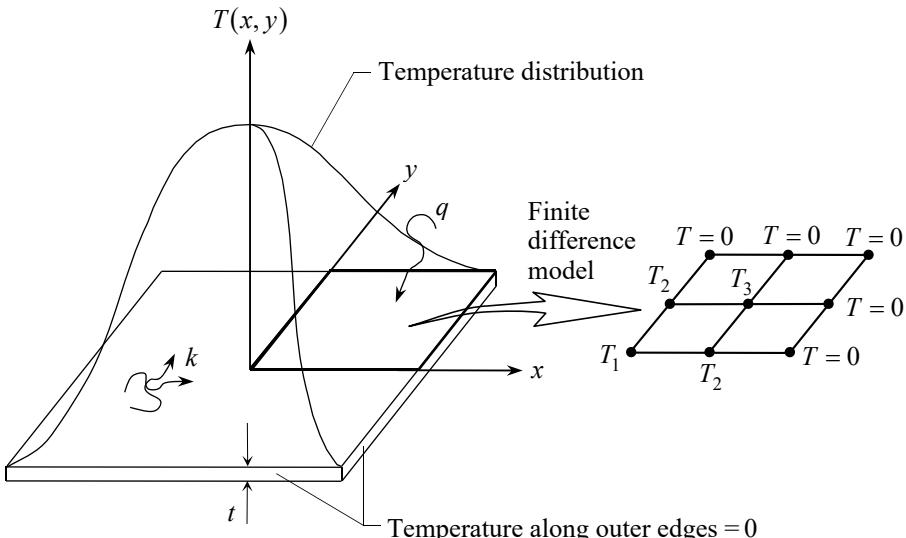


Figure 3.1 Temperature distribution on the square plate subjected to a surface heating and the finite difference model.

Approximate solution of the plate temperature can be obtained by using the finite difference technique. Details for the procedure of the finite difference technique will be explained in chapter 8. The technique starts from dividing the computational domain into a mesh with rectangular shape as shown in Fig. 3.1. Due to the symmetry of the temperature solution, only the upper-right quarter of the plate can be used for modeling in the analysis. If this quarter is divided into 2×2 intervals, the unknowns are the temperatures T_1 , T_2 and T_3 at the grid points as shown in the figure. The finite difference technique leads to the set of equations in the form

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.3)$$

The computed solutions at these grid points are $T_1 = 450$, $T_2 = 350$ and $T_3 = 275$. It should be noted that the computed temperatures at the grid points contain errors from the model discretization. Higher

solution accuracy can be obtained by modeling the plate with a finer mesh. Such the finer mesh, will lead to a set of more equations. For example, if the upper-right quarter of the plate is divided into 10×10 intervals, there will be a total 55 unknowns which are solved from 55 equations. Thus, in general, a system of equations can be written in the matrix form as,

$$\begin{matrix} [A] \{X\} \\ (n \times n) \end{matrix} = \begin{matrix} \{B\} \\ (n \times 1) \end{matrix} \quad (3.4)$$

where n is the number of equations, $[A]$ is a square matrix with the size of $(n \times n)$, the vector $\{X\}$ is a column matrix containing n unknowns and the vector $\{B\}$ is a column matrix with the known values. Definitions of matrices, their properties and manipulations are explained in appendix A.

In this chapter, several numerical methods for solving the system of equations in the form of Eq. (3.4) are explained. These methods are: (1) the Cramer's rule, (2) the Gauss elimination method, (3) the Gauss-Jordan method, (4) the matrix inversion method, (5) the LU decomposition method, (6) the Cholesky decomposition method, (7) the Jacobi iteration method, (8) the Gauss-Seidel iteration method, (9) the successive over-relaxation method, and (10) the conjugate gradient method.

3.2 Cramer's Rule

Cramer's rule is the method suitable for solving a system of few equations. The method solves for the solutions by finding determinants of the matrices. The unknown x_i in Eq. (3.4) is determined from

$$x_i = \frac{\det [A]_i}{\det [A]} \quad (3.5)$$

where $\det [A]$ is the determinant of matrix $[A]$, and $\det [A]_i$ is the determinant of same matrix $[A]$ but its column i is replaced by the vector $\{B\}$. The use of Cramer's rule is illustrated by the following examples.

Example 3.2 Use Cramer's rule to solve the following system of equations,

$$\begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 4 \\ -1 \end{Bmatrix} \quad (3.6)$$

Here, the matrix $[A]$ and vector $\{B\}$ are

$$[A] = \begin{bmatrix} 2 & 1 \\ 1 & -1 \end{bmatrix} ; \quad \{B\} = \begin{Bmatrix} 4 \\ -1 \end{Bmatrix}$$

By using the Cramer's rule as shown in Eq. (3.5), the unknowns x_1 and x_2 can be determined as follows

$$x_1 = \frac{\det [A]_1}{\det [A]} = \frac{\begin{vmatrix} 4 & 1 \\ -1 & -1 \end{vmatrix}}{\begin{vmatrix} 2 & 1 \\ 1 & -1 \end{vmatrix}} = \frac{-4+1}{-2-1} = \frac{-3}{-3} = 1$$

$$x_2 = \frac{\det[A]_2}{\det[A]} = \frac{\begin{vmatrix} 2 & 4 \\ 1 & -1 \\ 2 & 1 \\ 1 & -1 \end{vmatrix}}{\begin{vmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{vmatrix}} = \frac{-2-4}{-2-1} = \frac{-6}{-3} = 2$$

Example 3.3 Use the Cramer's rule to solve Eq. (3.3),

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix}$$

For this example, the matrix $[A]$ and vector $\{B\}$ are

$$[A] = \begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}; \quad \{B\} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix}$$

By using the Cramer's rule as shown in Eq. (3.5), the three unknowns are determined from

$$\begin{aligned} x_1 &= \frac{\det[A]_1}{\det[A]} = \frac{\begin{vmatrix} 400 & -4 & 0 \\ 400 & 4 & -2 \\ 400 & -2 & 4 \end{vmatrix}}{\begin{vmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{vmatrix}} \\ &= \frac{400(16-4) + 4(1,600+800) + 0}{4(16-4) + 4(-4-0) + 0} = \frac{14,400}{32} = 450 \\ x_2 &= \frac{\det[A]_2}{\det[A]} = \frac{\begin{vmatrix} 4 & 400 & 0 \\ -1 & 400 & -2 \\ 0 & 400 & 4 \end{vmatrix}}{\begin{vmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{vmatrix}} \\ &= \frac{4(1,600+800) - 400(-4-0) + 0}{4(16-4) + 4(-4-0) + 0} = \frac{11,200}{32} = 350 \\ x_3 &= \frac{\det[A]_3}{\det[A]} = \frac{\begin{vmatrix} 4 & -4 & 400 \\ -1 & 4 & 400 \\ 0 & -2 & 400 \end{vmatrix}}{\begin{vmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{vmatrix}} \\ &= \frac{4(1,600+800) + 4(-400-0) + 400(2-0)}{4(16-4) + 4(-4-0) + 0} = \frac{8,800}{32} = 275 \end{aligned}$$

The two examples above show that solutions of the system of equations can be obtained conveniently by using the Cramer's rule. However, it should be noted that the Cramer's rule is not used for solving a large system of equations. This is mainly because the method requires a large number of operations as compared to other methods. The number of operations required by the Cramer's rule to solve a set of n equations is $(n-1) \cdot (n+1)!$. If a set of equations contains only 10 equations, the method needs 360 million operations. Furthermore, about 10^{157} operations will be needed for solving a set of 100 equations. The method was thus never been used for solving practical problems that consist of a large number of equations. In the next topic, the Gauss elimination method will be presented. The method uses only $(4n^3 + 9n^2 - 7n)/6$ operations for solving a system of n equations. The method, thus, requires only about 700,000 operations to solve a set of 100 equations.

3.3 Gauss Elimination Method

The Gauss elimination method is one of the most popular methods for solving systems of equations generated from the computational procedures in scientific and engineering analyses. The Gauss elimination method for solving a set of equations is divided into two main steps as follows.

- (a) *Forward elimination.* For example, if the system of equations consists of 3 equations as,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \quad (3.7)$$

The first step of the method is to modify the matrix $[A]$ on the left-hand side of the equation so that all coefficients beneath its diagonal line are zero,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a'_{22} & a'_{23} \\ 0 & 0 & a''_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b'_2 \\ b''_3 \end{Bmatrix} \quad (3.8)$$

where the coefficients with prime symbols have different values from those shown in the original Eq. (3.7).

(b) *Back substitution.* With the equations in the form of Eq. (3.8) obtained from the first step, the unknowns x_i can be determined by using back substitution from the last to the first equation as follows,

$$\begin{aligned} x_3 &= b''_3/a''_{33} \\ x_2 &= (b'_2 - a'_{23}x_3)/a'_{22} \\ x_1 &= (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11} \end{aligned} \quad (3.9)$$

To understand the procedure of the Gauss elimination method more clearly, the set of three equations solved by the Cramer's rule in example 3.3 is repeated as presented below.

Example 3.4 Use the Gauss elimination method to solve the system of equations as shown in Eq. (3.3)

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

The above system of three equations can be written explicitly as

$$4x_1 - 4x_2 = 400 \quad (3.11a)$$

$$-x_1 + 4x_2 - 2x_3 = 400 \quad (3.11b)$$

$$-2x_2 + 4x_3 = 400 \quad (3.11c)$$

(a) *Forward elimination.* The method starts from dividing Eq. (3.11a) by the coefficient of x_1

$$x_1 - x_2 = 100$$

Then, multiplying it with the coefficient of x_1 from Eq. (3.11b)

$$-x_1 + x_2 = -100$$

The equation is subtracted from Eq. (3.11b) to obtain

$$3x_2 - 2x_3 = 500 \quad (3.11b')$$

By applying the same process to Eq. (3.11c),

$$-2x_2 + 4x_3 = 400 \quad (3.11c')$$

After one loop of forward elimination, the system of equations becomes

$$4x_1 - 4x_2 = 400 \quad (3.11a)$$

$$3x_2 - 2x_3 = 500 \quad (3.11b')$$

$$-2x_2 + 4x_3 = 400 \quad (3.11c')$$

The next loop of forward elimination starts from Eq. (3.11b') by dividing it with the coefficient of x_2

$$x_2 - \frac{2}{3}x_3 = \frac{500}{3}$$

and then multiplying it by the coefficient of x_2 from Eq. (3.11c')

$$-2x_2 + \frac{4}{3}x_3 = -\frac{1,000}{3}$$

By subtracting the equation above from Eq. (3.11c') to obtain

$$\frac{8}{3}x_3 = \frac{2,200}{3} \quad (3.11c'')$$

After the second loop of forward elimination, the system of equations now becomes

$$4x_1 - 4x_2 = 400 \quad (3.11a)$$

$$3x_2 - 2x_3 = 500 \quad (3.11b')$$

$$\frac{8}{3}x_3 = \frac{2,200}{3} \quad (3.11c'')$$

If the above equations are written in matrix form, the coefficients beneath the diagonal line of the square matrix on the left-hand side of the system equations are all zero as needed.

(b) *Back substitution.* By starting from the last equation, the solution x_3 can be determined. The computed solution x_3 is used to determine the values of x_2 and then x_1 , respectively,

$$x_3 = (2,200/3)(3/8) = 275$$

$$x_2 = (500 + 2(275))/3 = 350$$

$$x_1 = (400 + 4(350))/4 = 450$$

These values of x_1 , x_2 and x_3 are the solutions of Eq. (3.10).

Example 3.4 demonstrates that the Gauss elimination method has a systematic procedure and can be used to develop a computer program directly. The procedure as shown in example 3.4 can be generalized for solving a set of n equations as follows.

For a system of n equations,

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \quad (3.12a)$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \dots + a_{2n}x_n = b_2 \quad (3.12b)$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \dots + a_{3n}x_n = b_3 \quad (3.12c)$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \dots + a_{nn}x_n = b_n \quad (3.12n)$$

the two main steps of the forward elimination and back substitution are still applied to solve for the solutions.

Forward elimination The procedure starts from dividing the first equation (Eq. (3.12a)) by the coefficient of x_1

$$x_1 + \frac{a_{12}}{a_{11}}x_2 + \frac{a_{13}}{a_{11}}x_3 + \dots + \frac{a_{1n}}{a_{11}}x_n = \frac{b_1}{a_{11}}$$

By multiplying by the coefficient of x_1 from the second equation (Eq. (3.12b)),

$$a_{21}x_1 + a_{21}\frac{a_{12}}{a_{11}}x_2 + a_{21}\frac{a_{13}}{a_{11}}x_3 + \dots + a_{21}\frac{a_{1n}}{a_{11}}x_n = a_{21}\frac{b_1}{a_{11}}$$

and subtracting it from Eq. (3.12b) to eliminate the term associated with x_1 . Such manipulations yield,

$$\underbrace{\left(a_{22} - a_{21}\frac{a_{12}}{a_{11}}\right)}_{a'_{22}}x_2 + \underbrace{\left(a_{23} - a_{21}\frac{a_{13}}{a_{11}}\right)}_{a'_{23}}x_3 + \dots + \underbrace{\left(a_{2n} - a_{21}\frac{a_{1n}}{a_{11}}\right)}_{a'_{2n}}x_n = \underbrace{b_2 - a_{21}\frac{b_1}{a_{11}}}_{b'_2}$$

or,

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2 \quad (3.12b')$$

The process is repeated from Eq. (3.12c) to Eq. (3.12n) to obtain the system of equations in the form,

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \quad (3.13a)$$

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2 \quad (3.13b)$$

$$a'_{32}x_2 + a'_{33}x_3 + \dots + a'_{3n}x_n = b'_3 \quad (3.13c)$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots$$

$$a'_{n2}x_2 + a'_{n3}x_3 + \dots + a'_{nn}x_n = b'_n \quad (3.13n)$$

After the first loop of forward elimination, the terms associated with x_1 from the second equation onward are eliminated.

For the second loop of forward elimination, the terms associated with x_2 from Eqs. (3.13c) through (3.13n) are eliminated. The procedure starts from dividing Eq. (3.13b) by a'_{22} , multiplying it by the coefficient a'_{32} from Eq. (3.13c) and subtracting the result from Eq. (3.13c). These operations lead to a new system of equations in the form,

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \quad (3.14a)$$

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2 \quad (3.14b)$$

$$a''_{33}x_3 + \dots + a''_{3n}x_n = b''_3 \quad (3.14c)$$

$$\vdots \quad \vdots \quad \vdots$$

$$a''_{n3}x_3 + \dots + a''_{nn}x_n = b''_n \quad (3.14n)$$

The same procedure is repeated until the loop number $(n-1)$ is reached and the system of equations finally becomes,

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \dots + a_{1n}x_n = b_1 \quad (3.15a)$$

$$a'_{22}x_2 + a'_{23}x_3 + \dots + a'_{2n}x_n = b'_2 \quad (3.15b)$$

$$a''_{33}x_3 + \dots + a''_{3n}x_n = b''_3 \quad (3.15c)$$

$$\vdots \quad \vdots$$

$$a_{nn}^{(n-1)}x_n = b_n^{(n-1)} \quad (3.15n)$$

The prime symbols and the values in the superscripts represent the number of the computational loop needed for the forward elimination process.

Back substitution From Eq. (3.15), the unknown x_n can be determined directly from the last equation, Eq. (3.15n), as

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}} \quad (3.16a)$$

Then, the unknowns $x_{n-1}, x_{n-2}, \dots, x_2, x_1$ can be obtained by using back substitution as follows. The computed value of x_n is substituted into the $(n-1)^{th}$ equation to solve for the unknown x_{n-1} . Then,

the computed values of x_n and x_{n-1} are substituted into the $(n-2)^{th}$ equation to solve for the unknown x_{n-2} . The same procedure is repeated to determine the remaining values of the unknowns x_i . Determination of the unknowns x_i , $i = n-1, \dots, 1$ can be obtained using the expression,

$$x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^n a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}} \quad (3.16b)$$

where subscript i represents the i^{th} equation in Eq. (3.15).

Procedures of the Gauss elimination method for solving the system of equations as explained from Eq. (3.12) to (3.16) are used to develop a computer program as shown in Fig. 3.2. Table 3.1 shows the input data generated from example 3.4 together with the computed solutions from the program.

```
% Program NGElim
% A program for solving a set of linear
% simultaneous equations [A]{X} = {B} by
% using Naive Gauss elimination method.
% Read input data file containing number
% of eqs with matrix [A] and vector {B}:
fid = fopen('input.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [n+1 n]); a = a.';
b = squeeze(a(:, n+1));
a = squeeze(a(:, 1:n));
fclose(fid);
x = NGauss(n, a, b);
% Print out the computed solution of {X}:
fid = fopen('output.dat', 'w');
fprintf(fid, ...
'\n Equation No.      Solution x \n');
fprintf( ...
'\n Equation No.      Solution x \n');
for i = 1:n
    fprintf(fid, ' %8d %22.6e\n', i, x(i));
    fprintf( ' %8d %22.6e\n', i, x(i));
end
fclose(fid);

function x = NGauss(n, a, b)
% Forward elimination: Perform according
% to order of 'Prime' from 1 to n-1:
for ip = 1:n-1
    % Loop over each equation starting from
    % the one that corresponds with the
    % order of 'Prime' plus one:
    for ie = ip+1:n
        ratio = a(ie,ip)/a(ip,ip);
        % Compute new coeff. of the eq. considered:
        for ic = ip+1:n
            a(ie,ic) = a(ie,ic) - ratio*a(ip,ic);
        end
        b(ie) = b(ie) - ratio*b(ip);
    end
    % Set coeff. on lower left portion to zero:
    for ie = ip+1:n
        a(ie,ip) = 0.;
    end
end
% Back substitution:
% Compute solution of the last equation:
x(n) = b(n)/a(n,n);
% Compute solutions from equation n-1 to 1:
for ie = n-1:-1:1
    sum = 0.;
    for ic = ie+1:n
        sum = sum + a(ie,ic)*x(ic);
    end
    x(ie) = (b(ie) - sum)/a(ie,ie);
end
```

Figure 3.2 Computer program for solving a system of equations by the Gauss elimination method.

Table 3.1 Example of input data from Eq. (3.10) for using with the program in Fig. 3.2 and the computed solutions by the Gauss elimination method.

Input data from Eq. (3.10)

3			
4.	-4.	0.	400.
-1	4.	-2.	400.
0.	-2.	4.	400.

Computed solutions

Equation No.	Solution x
1	.450000E+03
2	.350000E+03
3	.275000E+03

3.4 Problems of Gauss Elimination Method

Example 3.4 shows that the process in the Gauss elimination method is simple and easy to understand. The corresponding computer program can be developed easily as shown in Fig. 3.2. However, the Gauss elimination method explained in the preceding section may have some pitfalls as will be explained in this section.

3.4.1 Division by zero.

The system of equations used in example 3.4 is

$$4x_1 - 4x_2 = 400 \quad (3.11a)$$

$$-x_1 + 4x_2 - 2x_3 = 400 \quad (3.11b)$$

$$-2x_2 + 4x_3 = 400 \quad (3.11c)$$

In the process of forward elimination, Eq. (3.11a) is first divided by the coefficient x_1 that has the value of 4. If Eq. (3.11a) is interchanged with Eq. (3.11c) so that the system of equations becomes

$$-2x_2 + 4x_3 = 400 \quad (3.11c)$$

$$-x_1 + 4x_2 - 2x_3 = 400 \quad (3.11b)$$

$$4x_1 - 4x_2 = 400 \quad (3.11a)$$

The coefficient of x_1 in the first equation is now zero. Division by zero thus creates problem in the forward elimination process. For general problems, it is possible that the coefficient $a_{ii}^{(i-1)}$ in Eq. (3.14) may be zero or close to zero. Small values of $a_{ii}^{(i-1)}$ can also lead to inaccurate solutions. The techniques to avoid these problems will be explained in section 3.5.

3.4.2 Round-off error.

Another problem that can occur in the process of the Gauss elimination method is the round-off error. Values are stored in computer with a limit amount of significant figures during computation. Numerical operations in the computational process of the Gauss elimination method can produce solutions with round-off error. The round-off error propagates as the number of operations increases. Such error can be understood clearly by studying the following example.

Example 3.5 Given the system of 2 equations as,

$$\begin{bmatrix} 2.000112 & 1.414214 \\ 1.414214 & 1.000102 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 0.521471 \\ 0.232279 \end{Bmatrix} \quad (3.17)$$

If the computer program as shown in Fig. 3.2 is used and executed on a typical computer, the computed solutions are $x_1 = 613.0448$ and $x_2 = -866.6558$. But if a calculator with 6 significant figures is used, the system of equations, Eq. (3.17), becomes

$$\begin{bmatrix} 2.00011 & 1.41421 \\ 1.41421 & 1.00010 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 0.521471 \\ 0.232279 \end{Bmatrix} \quad (3.18)$$

With the forward elimination process, Eq. (3.18) is changed to

$$\begin{bmatrix} 2.00011 & 1.41421 \\ 0 & 0.167263 \times 10^{-3} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 0.521471 \\ -0.136435 \end{Bmatrix}$$

After performing the back substitution, the computed solutions are $x_1 = 577.008$ and $x_2 = -815.691$. These solutions have an error of 6% as compared to the true solutions of Eq. (3.17). It is noted, in addition, that such a large error occurs from the fact that the square matrix on the left-hand side of the equation is an ill-conditioned matrix.

3.4.3 Ill-conditioned system.

A square matrix on the left-hand side of Eq. (3.17) is an ill-conditioned matrix because a small change of its coefficients in the matrix

$$[A] = \begin{bmatrix} 2.000112 & 1.414214 \\ 1.414214 & 1.000102 \end{bmatrix} \quad (3.19)$$

can cause a large change in the computed solutions. A matrix can be classified as an ill-conditioned matrix if its determinant is close to zero or very small as compared to its coefficients. In this example, the determinant of matrix $[A]$ is

$$\begin{aligned} \det[A] &= (2.000112)(1.000102) - (1.414214)(1.414214) \\ &= 0.000315 \end{aligned}$$

which is very small as compared to the matrix coefficients. For a practical engineering problem, a system of equations may consist of hundred thousand equations so that the size of matrix $[A]$ is very large. Finding its determinant is not practical because a large computational time is required. It is noted that, however, most engineering problems lead to the well-conditioned systems of equations. For example, the determinant of matrix $[A]$ is 32 in the example of heat transfer on the metal plate as shown in example 3.1.

In summary, the matrix $[A]$ is an ill-conditioned matrix if one or more of the following conditions exist:

- (a) A small change in matrix $[A]$ coefficients causes a large change in the computed solutions.
- (b) Values of the diagonal coefficients in matrix $[A]$ are small as compared to the off-diagonal ones.
- (c) The determinant of matrix $[A]$ is close to zero or the product of $(\det[A]) \cdot (\det[A]^{-1})$ is far from unity.
- (d) The product of $[A][A]^{-1}$ is somewhat different from the identity matrix.

3.5 Improved Gauss Elimination Method

As explained in the preceding section, some difficulties may be encountered while using the Gauss elimination method. One of the major difficulties is the division of coefficients by zero or values that are close to zero. Such difficulty can be avoided or alleviated by using the techniques of pivoting and scaling as described below.

3.5.1 Pivoting

If any coefficient a_{ii} along the diagonal line of the $[A]$ matrix is zero, the method of Gauss elimination method can not proceed. Or if such coefficients are close to zero, the method will produce solutions with error. Inaccurate solutions that occur in the later case can be seen clearly by considering the following example.

Example 3.6 Solve the following system of equations by using the standard Gauss elimination method,

$$\begin{bmatrix} 0.0003 & 3.0000 \\ 1.0000 & 1.0000 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 2.0001 \\ 1.0000 \end{Bmatrix} \quad (3.20)$$

The exact solutions for the set of the two equations above are $x_1 = 1/3$ and $x_2 = 2/3$. It is noted that the value of the diagonal coefficient in the first equation is very small (0.0003) as compared to the other coefficients (3.0000 and 1.0000). After performing the forward elimination, the system of equations becomes

$$\begin{bmatrix} 0.0003 & 3.0000 \\ 0 & -9,999 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 2.0001 \\ -6,666 \end{Bmatrix} \quad (3.21)$$

Then, by using the back substitution, the two solutions are obtained as

$$x_2 = 2/3$$

and

$$x_1 = (2.0001 - 3x_2)/0.0003$$

The accuracy of the computed solution x_1 depends on the amount of the significant figures used during computation. For example, if a calculator with only 5 significant figures is used, the computed solutions are,

$$x_2 = 0.66667 \quad \text{and} \quad x_1 = 0.30000 \quad (3.22)$$

The error of the computed solution x_1 is about 10%.

If the two equations in Eq. (3.20) are switched, i.e., their orders are interchanged such that,

$$\begin{bmatrix} 1.0000 & 1.0000 \\ 0.0003 & 3.0000 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1.0000 \\ 2.0001 \end{Bmatrix} \quad (3.23)$$

Then, by performing the forward elimination,

$$\begin{bmatrix} 1.0000 & 1.0000 \\ 0 & 2.9997 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1.0000 \\ 1.9998 \end{Bmatrix}$$

and back substitution, the computed solutions are,

$$x_2 = 0.66667 \quad \text{and} \quad x_1 = 0.33333 \quad (3.24)$$

In this case, the error of the computed solution x_1 reduces to 0.001%.

The example shows that accuracy of the solutions obtained from the Gauss elimination method depends on the values of the coefficients along the diagonal line of the matrix $[A]$. To obtain solutions with a higher accuracy, the orders of the equations must be changed such that the magnitudes of the coefficients along the matrix diagonal line are large as compared to its off-diagonal terms. To solve a set of n equations as shown in Eq. (3.12), the orders of equations must be rearranged. This is done by selecting the equation that has the largest coefficient and moving it up while performing the forward elimination process. As an example, before performing the third loop in the forward elimination as shown in Eq. (3.14), the equation within Eqs. (3.14c) – (3.14n) that has the largest coefficient of x_3 is

selected and interchanged with the former Eq. (3.14c). Such process is called partial pivoting and presented in the subroutine (or function in MATLAB) PIVOT in the computer program in Fig. 3.3.

3.5.2 Scaling

The matrix $[A]$ in a system of equations generated from an engineering problem may consist of coefficients with a large difference in magnitudes. For example, a system of equations that occurs in the high-speed compressible flow analysis around the space shuttle in Fig. 1.3 may consist of unknowns of the density, velocity components and temperature of the flow field. The matrix $[A]$ of such problem consists of coefficients that are quite different in magnitudes. Large differences in the magnitudes of these coefficients create solution error as demonstrated in the following example.

Example 3.7 Solve the following system of equations by the Gauss elimination method,

$$\begin{bmatrix} 2 & 100,000 \\ 1 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 100,000 \\ 2 \end{Bmatrix} \quad (3.25)$$

The exact solutions of the above set of equations are $x_1 = 1.00002$ and $x_2 = 0.99998$.

If the standard Gauss elimination method is used, the forward elimination process yields,

$$\begin{bmatrix} 2 & 100,000 \\ 0 & -49,999 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 100,000 \\ -49,999 \end{Bmatrix} \quad (3.26)$$

After performing the back substitution, the computed solutions with three significant figures are,

$$x_2 = 1.00 \quad \text{and} \quad x_1 = 0.00 \quad (3.27)$$

for which the error of x_1 is 100%.

But, if the scaling process is first applied by dividing each equation by the largest value of all coefficients in that equation, Eq. (3.25) becomes,

$$\begin{bmatrix} 0.00002 & 1 \\ 1 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \end{Bmatrix} \quad (3.28)$$

Then, by applying the pivoting technique as explained in the preceding section,

$$\begin{bmatrix} 1 & 1 \\ 0.00002 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 1 \end{Bmatrix}$$

With the forward elimination, the set of equations becomes,

$$\begin{bmatrix} 1 & 1 \\ 0 & 0.99998 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 0.99996 \end{Bmatrix}$$

After performing the back substitution, the computed solutions are

$$x_2 = 1.00 \quad \text{and} \quad x_1 = 1.00 \quad (3.29)$$

This example demonstrates that the scaling technique can increase accuracy of the computed solutions. A subroutine for performing the scaling technique can also be developed easily. Figure 3.3 shows a typical function `scale` that can be included into the standard Gauss elimination program.

Example 3.8 Modify the Gauss elimination computer program as shown in Fig. 3.2 by including the capability of pivoting and scaling. Check accuracy of the computed solutions obtained from the modified computer program by using Eq. (3.10) in example 3.4.

The functions `Scale` and `Pivot` as shown in Fig. 3.3 are called by the function `Gauss` before and after the forward elimination loop. Details of each subroutine are presented in Fig. 3.3. Table 3.2 shows the computed solutions of Eq. (3.10) generated from the modified program.

```
% Program GElim
% A program for solving a set of linear
% simultaneous equations [A]{X} = {B} by
% using Gauss elimination method.
% Read input data file containing number
% of eqs with matrix [A] and vector (B):
fid = fopen('input.dat', 'r');
n = fsccanf(fid, '%f', 1);
a = fsccanf(fid, '%f', [n+1 n]); a = a.';
b = squeeze(a(:, n+1));
a = squeeze(a(:, 1:n));
fclose(fid);
x = Gauss(n, a, b);
% Print out the computed solution of {X}:
fid = fopen('output.dat', 'w');
fprintf(fid, ...
'\n Equation No. Solution x \n');
fprintf( ...
'\n Equation No. Solution x \n');
for i = 1:n
    fprintf(fid, '%8d %22.6e\n', i, x(i));
    fprintf( '%8d %22.6e\n', i, x(i));
end
fclose(fid);

function x = Gauss(n, a, b)
% Perform scaling:
[a, b] = Scale(n, a, b);
% Forward elimination: Perform according
% to order of 'Prime' from 1 to n-1:
for ip = 1:n-1
    % Perform partial pivoting:
    [a, b] = Pivot(n, a, b, ip);
    % Loop over each equation starting from
    % the one that corresponds with the
    % order of 'Prime' plus one:
    for ie = ip+1:n
        ratio = a(ie,ip)/a(ip,ip);
        % Compute new coeff. of the eq. considered:
        for ic = ip+1:n
            a(ie,ic) = a(ie,ic) - ratio*a(ip,ic);
        end
        b(ie) = b(ie) - ratio*b(ip);
    end
    % Set coeff. on lower left portion to zero:
    for ie = ip+1:n
        a(ie,ip) = 0.;
    end
end
% Back substitution:
x(n) = b(n)/a(n,n);
% Compute solutions from equation n-1 to 1:
for ie = n-1:-1:1
    sum = 0.;
    for ic = ie+1:n
        sum = sum + a(ie,ic)*x(ic);
    end
    x(ie) = (b(ie) - sum)/a(ie,ie);
end

function [a, b] = Scale(n, a, b)
for ie = 1:n
    big = abs(a(ie,1));
    for ic = 2:n
        amax = abs(a(ie,ic));
        if amax > big
            big = amax;
        end
    end
    for ic = 1:n
        a(ie,ic) = a(ie,ic)/big;
    end
    b(ie) = b(ie)/big;
end

function [a, b] = Pivot(n, a, b, ip)
jp = ip; big = abs(a(ip,ip));
for i = ip+1:n
    amax = abs(a(i,ip));
    if amax > big
        big = amax; jp = i;
    end
end
if jp ~= ip
    for j = ip:n
        dumy = a(jp,j);
        a(jp,j) = a(ip,j);
        a(ip,j) = dumy;
    end
    dumy = b(jp);
    b(jp) = b(ip);
    b(ip) = dumy;
end
```

Figure 3.3 Gauss elimination computer program after including the capability of scaling and pivoting.

Table 3.2 Example of input data file for Eq. (3.10) for the modified program and its computed solutions.

Input data for Eq. (3.10)

3			
4.	-4.	0.	400.
-1.	4.	-2.	400.
0.	-2.	4.	400.

Computed solutions

Equation No.	Solution x
1	.450000E+03
2	.350000E+03
3	.275000E+03

3.5.3 Tridiagonal system

The method of Gauss elimination can be used effectively to solve a special form of the system of equations,

$$\begin{matrix} [A] \{X\} \\ (n \times n) (n \times 1) \end{matrix} = \begin{matrix} \{B\} \\ (n \times 1) \end{matrix} \quad (3.4)$$

There are many engineering problems that yield the matrix $[A]$ with the non-zero terms only along the three main diagonal lines,

$$[A] = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & a_{n,n-1} & a_{n,n} \end{bmatrix} \quad (3.30)$$

In this case, the system Eq. (3.4) is written in full as,

$$\begin{bmatrix} b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ a_{n-1} & b_{n-1} & c_{n-1} \\ a_n & b_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \\ \cdot \\ \cdot \\ \cdot \\ d_{n-1} \\ d_n \end{Bmatrix} \quad (3.31)$$

The Gauss elimination method can be applied to solve such system of equations by eliminating the coefficients a_2, a_3, \dots, a_n in the forward elimination process and then performing the back substitution in order to obtain the solutions.

Example 3.9 Use the Gauss elimination method to solve the set of tridiagonal system in Eq. (3.10),

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

The process starts from forward elimination by dividing the first equation by b_1 , multiplying with a_2 , and subtracting it from the second equation to obtain the new value of b'_2 and d'_2 as,

$$\begin{aligned} b'_2 &= b_2 - \frac{a_2}{b_1} c_1 = 4 - \frac{(-1)}{4} (-4) = 3 \\ d'_2 &= d_2 - \frac{a_2}{b_1} d_1 = 400 - \frac{(-1)}{4} (400) = 500 \end{aligned}$$

It is noted that the value of c_2 is not altered because the coefficient above it is zero.

The forward elimination is then applied to the third equation to obtain,

$$\begin{bmatrix} 4 & -4 & 0 \\ 0 & 3 & -2 \\ 0 & 0 & 8 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 500 \\ 2,200 \end{Bmatrix} \quad (3.32)$$

With back substitution starting from the last equation, the solutions of x_3 , x_2 and x_1 , are determined,

$$x_3 = 275 ; \quad x_2 = 350 ; \quad x_1 = 450$$

For a tridiagonal system of equations, a function program can be developed so that only the non-zero coefficients of a , b and c are stored as the three vectors for minimizing computer memory. The Gauss elimination process is performed in the program by using these three vectors as shown in Fig. 3.4. It is noted that a_i , b_i , c_i and d_i , $i = 1, 1, \dots, n$, are defined in Eq. (3.31).

```
function x = TriDg(a, b, c, d, n)
%
% Solve tridiagonal system of n equations
% Perform forward elimination. Note that
% a(i), b(i), c(i) and d(i), i=1,2,...,n
% are defined in Eq. (3.31).
%
for i = 2:n
    a(i) = a(i)/b(i-1);
    b(i) = b(i) - a(i)*c(i-1);
    d(i) = d(i) - a(i)*d(i-1);
end
%
% Perform backward substitution:
%
x(n) = d(n)/b(n)
for i = n-1:-1:1
    x(i) = (d(i) - c(i)*x(i+1))/b(i);
end
```

Figure 3.4 A function program for solving tridiagonal system of equations.

3.6 Gauss-Jordan Method

The Gauss-Jordan method is an extension of the Gauss elimination method for solving a set of equations,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \quad (3.33)$$

so that, after performing the elimination, it reduces into the form of,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1^* \\ b_2^* \\ b_3^* \end{Bmatrix} \quad (3.34)$$

Because the square matrix on the left-hand side of Eq. (3.34) is an identity matrix, thus the equation gives the solutions directly as,

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1^* \\ b_2^* \\ b_3^* \end{Bmatrix}$$

Example 3.10 Solve Eq. (3.10) again by using the Gauss-Jordan method.

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

By using Gauss elimination method as explained in example 3.4, the resulting matrix after applying forward elimination is,

$$\begin{bmatrix} 4 & -4 & 0 \\ 0 & 3 & -2 \\ 0 & 0 & 8/3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 500 \\ 2,200/3 \end{Bmatrix}$$

All equations are modified so that the coefficients along the main diagonal line of matrix $[A]$ are equal to one,

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2/3 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 100 \\ 500/3 \\ 275 \end{Bmatrix}$$

The forward elimination is applied to eliminate the coefficient of x_2 in the first equation,

$$\begin{bmatrix} 1 & 0 & -2/3 \\ 0 & 1 & -2/3 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 800/3 \\ 500/3 \\ 275 \end{Bmatrix}$$

Similarly, the coefficient of x_3 in the first equation is eliminated,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2/3 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 450 \\ 500/3 \\ 275 \end{Bmatrix}$$

After the coefficient of x_3 in the second equation is eliminated, the system of equations becomes,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 450 \\ 350 \\ 275 \end{Bmatrix}$$

Thus, the solutions of the system of equations are,

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 450 \\ 350 \\ 275 \end{Bmatrix}$$

The above example illustrates that the Gauss-Jordan method leads to the solutions of the system of equations directly. It is noted that the difficulties that arise by using the Gauss elimination method also occur in the Gauss-Jordan method. The techniques of pivoting and scaling are thus needed to apply. In practice, the method is not used for solving a set of equations. This is because the method requires a large number of operations in the order of $(n^3 + n^2 - n)$, where n is the number of equations. Such the number of operations is greater than that required by the Gauss elimination method. However, the

Gauss-Jordan method offers an advantage for finding the inverse of a matrix conveniently as will be explained in the next section.

3.7 Matrix Inversion Method

The Gauss-Jordan method can be used to find the inverse of a matrix. If a matrix with the size of (3×3) , for example, is considered,

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad (3.35)$$

The multiplication of the matrix $[A]$ and its matrix inverse $[A]^{-1}$ leads to,

$$\underset{(3 \times 3)}{[A]} \underset{(3 \times 3)}{[A]^{-1}} = \underset{(3 \times 3)}{[I]} \quad (3.36)$$

where $[I]$ is the identity matrix. If the matrix inverse consists of three column vectors as,

$$\underset{(3 \times 3)}{[A]^{-1}} = \left[\begin{array}{c|c|c} \{X\}_1 & \{X\}_2 & \{X\}_3 \\ \hline (3 \times 1) & (3 \times 1) & (3 \times 1) \end{array} \right] \quad (3.37)$$

then, by substituting it into Eq. (3.36),

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix}_1 & \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix}_2 & \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.38)$$

Equation (3.38) can be arranged into the three systems of equations as,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (3.39)$$

These three systems of equations have the same square matrix $[A]$ on left-hand side but with three different vectors on the right-hand side. The Gauss-Jordan method can be applied to these three systems of equations simultaneously as shown in the example below.

Example 3.11 Solve Eq. (3.10) again by using the matrix inversion method.

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 400 \\ 400 \\ 400 \end{bmatrix} \quad (3.10)$$

In this example, the matrix $[A]$ is

$$[A] = \begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}$$

To determine the matrix inverse $[A]^{-1}$, the following systems of equations are solved,

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 0 \\ 0 \end{Bmatrix}, \begin{Bmatrix} 0 \\ 1 \\ 0 \end{Bmatrix}, \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \quad (3.40)$$

By applying the Gauss-Jordan method as explained in section 3.6, the systems of equations below are obtained,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 3/8 \\ 1/8 \\ 1/16 \end{Bmatrix}, \begin{Bmatrix} 1/2 \\ 1/2 \\ 1/4 \end{Bmatrix}, \begin{Bmatrix} 1/4 \\ 1/4 \\ 3/8 \end{Bmatrix} \quad (3.41)$$

Thus, the matrix inverse is,

$$[A]^{-1} = \begin{bmatrix} 3/8 & 1/2 & 1/4 \\ 1/8 & 1/2 & 1/4 \\ 1/16 & 1/4 & 3/8 \end{bmatrix} \quad (3.42)$$

Then, the solutions of Eq. (2.10) can be determined directly as,

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{bmatrix} 3/8 & 1/2 & 1/4 \\ 1/8 & 1/2 & 1/4 \\ 1/16 & 1/4 & 3/8 \end{bmatrix} \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} = \begin{Bmatrix} 450 \\ 350 \\ 275 \end{Bmatrix} \quad (3.43)$$

From this example, the matrix inversion method can be used for finding the matrix inverse and solving for solutions of the system of equations. However, the number of operations required by this method is quite large (about $2n^3$) as compared to those needed by the other methods. Thus, the matrix inversion method is not employed in solving systems of equations for practical problems. However, the method can be used for testing the ill-conditioning of the matrix $[A]$ when its size is not too large as follows.

- (a) Apply the scaling technique as explained in section 3.5.2 on matrix $[A]$ and then find the matrix inverse $[A]^{-1}$. If coefficients in the matrix inverse $[A]^{-1}$ are large as compared to the coefficients in the original matrix $[A]$, the matrix is considered as an ill-conditioned matrix.
- (b) Multiply matrix $[A]$ by $[A]^{-1}$ to obtain a matrix. If that matrix is not close to the identity matrix $[I]$, the matrix $[A]$ is considered as an ill-conditioned matrix.
- (c) Determine the matrix inverse $[A]^{-1}$ and find the matrix inverse of the matrix $[A]^{-1}$ again. If the resulting matrix is not close to the original matrix $[A]$, the matrix is considered as an ill-conditioned matrix.

3.8 Solving System of Linear Equations by MATLAB

In MATLAB, the slash symbol (/) is used for division. For example, $2/3$ has the meaning of dividing 2 by 3. If this symbol is used for the matrix operation such as $[U]/[V]$, it means that the matrix $[U]$ is multiplied by the inverse of matrix $[V]$, i.e., $[U] \times [V]^{-1}$. The backslash symbol (\) in MATLAB

is called the left division and is used for solving a system of equations. For example, $[U] \setminus [V]$ means that the inverse of matrix $[U]$ is multiplied by matrix $[V]$, i.e., $[U]^{-1} \times [V]$.

For a system of equations in the form,

$$[A]\{X\} = \{B\} \quad (3.4)$$

The vector $\{X\}$, which contains solutions of the system of equations, is obtained by multiplying the inverse of matrix $[A]$ on both sides of the equations,

$$\begin{aligned}[A]^{-1}[A]\{X\} &= [A]^{-1}\{B\} \\ [I]\{X\} &= [A]^{-1}\{B\} \\ \{X\} &= [A]^{-1}\{B\}\end{aligned}$$

The vector $\{X\}$ can be obtained easily by using the backslash symbol or left division between the matrices $[A]$ and $\{B\}$ as shown in the following example.

Example 3.11 Solve the following system of equations by using MATLAB,

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

The procedures for obtaining the solutions of the vector $\{X\}$ are as follows,

```
>> A = [4 -4 0; -1 4 -2; 0 -2 4];
>> B = [400; 400; 400];
>> x = A\B
x =
    450.0000
    350.0000
    275.0000
```

In general, MATLAB uses the Gauss elimination method with partial pivoting for solving a system of equations when the left division is employed. Other solution methods, such as the LU decomposition and Cholesky decomposition, can also be selected. These methods are explained in details in the following sections.

3.9 LU Decomposition Method

One of the popular methods for solving a large system of equations is the LU decomposition method. The basic idea of this method is to decompose the matrix $[A]$ as the product of the two matrices $[L]$ and $[U]$. The matrix $[L]$ contains non-zero coefficients on the lower left portion of the matrix, while all other coefficients on the upper right portion are zero. The matrix $[U]$ contains the zero and non-zero coefficients similar to the matrix $[L]$ but in the opposite pattern. Procedures of the LU decomposition method can be explained by considering the system of equations,

$$[A]\{X\} = \{B\} \quad (3.4)$$

The first step is to decompose the matrix $[A]$ into the product of matrices $[L]$ and $[U]$ as,

$$[A] = [L][U] \quad (3.44)$$

For example, if the size of the matrix $[A]$ is (3x3), then

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} \quad (3.45)$$

where the matrix $[L]$ contains the non-zero coefficients only in the lower left portion. The matrix $[U]$ contains the non-zero coefficients only in the upper right portion with the values of one along the main diagonal line. By substituting Eq. (3.45) into Eq. (3.43),

$$[L] \underbrace{[U]\{X\}}_{\{Y\}} = \{B\} \quad (3.46)$$

Equation (3.46) can be written as,

$$[L]\{Y\} = \{B\} \quad (3.47)$$

which can be solved by the forward substitution to obtain the values in the vector $\{Y\}$. Then, the values in the vector $\{X\}$ can be determined by the back substitution from,

$$[U]\{X\} = \{Y\} \quad (3.48)$$

From the above explanation, the LU decomposition method for solving a system of equations $[A]\{X\} = \{B\}$ consists of the three main steps:

- (a) Decompose the matrix $[A]$ into the two matrices $[L]$ and $[U]$,
- (b) Solve the equations $[L]\{Y\} = \{B\}$ by forward substitution to obtain the vector $\{Y\}$,
- (c) Then, solve the equations $[U]\{X\} = \{Y\}$ by back substitution to obtain the vector $\{X\}$ which contains the solutions of the system of equations.

The most time consuming process is the decomposition of the matrix $[A]$ into the two matrices $[L]$ and $[U]$. In many engineering problems, the matrix $[A]$ and the vector $\{B\}$ have their physical meanings. As an example of the stresses that occur in a bridge structure from the load of moving vehicles, the matrix $[A]$ represents the stiffness of the bridge structure while the vector $\{B\}$ contains the load of the moving vehicles. The vector $\{B\}$ changes with different loadings from the number of vehicles, while the matrix $[A]$ representing the bridge structure stiffness remains the same. This means, to solve for the solutions under different loadings, the matrix $[A]$ can be decomposed only once into the product of $[L][U]$. The vector $\{B\}$ is changed according to the loading conditions. Thus, the LU decomposition method offers an advantage in reducing the computational time where many sets of the solutions $\{X\}$ are needed from the different vectors $\{B\}$.

When the matrix $[A]$ is decomposed into the matrices $[L]$ and $[U]$ as shown in Eq. (3.45), the coefficients along the main diagonal line of the matrix $[L]$ can be any value while those along the main diagonal line of the matrix $[U]$ are set as one. With such setting, the method is called the Crout

decomposition. On the other hand, if the coefficients along the main diagonal line of the matrix $[L]$ are set to be one, the method is called the Doolittle decomposition. Either the decomposition method can be used for solving the system of equations. Details of LU decomposition based on the Crout method are presented in the following example.

Example 3.13 Use the LU decomposition method to solve of the solutions of the system of equations,

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

Here, the matrix $[A]$ is

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}$$

The matrix $[A]$ can be decomposed into the matrices $[L]$ and $[U]$ by first writing,

$$\begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \quad (3.49)$$

From Eq. (3.49), the coefficients of the matrices $[L]$ and $[U]$ can be determined as follows:

$$\ell_{11} = a_{11} = 4 \quad (3.50a)$$

$$\ell_{21} = a_{21} = -1 \quad (3.50b)$$

$$\ell_{31} = a_{31} = 0 \quad (3.50c)$$

$$\ell_{11} u_{12} = a_{12} \rightarrow u_{12} = a_{12}/\ell_{11} = -1 \quad (3.51a)$$

$$\ell_{11} u_{13} = a_{13} \rightarrow u_{13} = a_{13}/\ell_{11} = 0 \quad (3.51b)$$

$$\ell_{21} u_{12} + \ell_{22} = a_{22} \rightarrow \ell_{22} = a_{22} - \ell_{21} u_{12} = 3 \quad (3.52a)$$

$$\ell_{31} u_{12} + \ell_{32} = a_{32} \rightarrow \ell_{32} = a_{32} - \ell_{31} u_{12} = -2 \quad (3.52b)$$

$$\ell_{21} u_{13} + \ell_{22} u_{23} = a_{23} \rightarrow u_{23} = \frac{a_{23} - \ell_{21} u_{13}}{\ell_{22}} = -\frac{2}{3} \quad (3.53a)$$

$$\ell_{31} u_{13} + \ell_{32} u_{23} + \ell_{33} = a_{33} \rightarrow \ell_{33} = a_{33} - \ell_{31} u_{13} - \ell_{32} u_{23} = \frac{8}{3} \quad (3.53b)$$

Therefore,

$$[A] = [L][U] = \begin{bmatrix} 4 & 0 & 0 \\ -1 & 3 & 0 \\ 0 & -2 & 8/3 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2/3 \\ 0 & 0 & 1 \end{bmatrix}$$

Then, Eq. (3.10) can be written in the form,

$$\underbrace{\begin{bmatrix} 4 & 0 & 0 \\ -1 & 3 & 0 \\ 0 & -2 & 8/3 \end{bmatrix}}_{[L]} \underbrace{\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2/3 \\ 0 & 0 & 1 \end{bmatrix}}_{[U]} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\{X\}} = \underbrace{\begin{bmatrix} 400 \\ 400 \\ 400 \end{bmatrix}}_{\{B\}}$$

$\underbrace{\quad\quad\quad}_{\{Y\}}$

(3.54)

The system of equations $[L]\{Y\} = \{B\}$ is first solved by forward substitution to give the unknowns in vector $\{Y\}$,

$$\begin{bmatrix} 4 & 0 & 0 \\ -1 & 3 & 0 \\ 0 & -2 & 8/3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 400 \\ 400 \\ 400 \end{bmatrix}$$

$$\begin{aligned} 4y_1 &= 400 \rightarrow y_1 = 100 \\ -y_1 + 3y_2 &= 400 \rightarrow y_2 = 500/3 \\ -2y_2 + \frac{8}{3}y_3 &= 400 \rightarrow y_3 = 275 \end{aligned}$$

Then, the system of equations $[U]\{X\} = \{Y\}$ is used to determine the unknowns in vector $\{X\}$ by back substitution as follows,

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -2/3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 100 \\ 500/3 \\ 275 \end{bmatrix}$$

$$\begin{aligned} x_3 &= 275 \rightarrow x_3 = 275 \\ x_2 - \frac{2}{3}x_3 &= 500/3 \rightarrow x_2 = 350 \\ x_1 - x_2 &= 100 \rightarrow x_1 = 450 \end{aligned}$$

so that all solutions of Eq. (3.10) are obtained.

The procedures for determining coefficients in the matrices $[L]$ and $[U]$ from the matrix $[A]$ as presented in example 3.13 can be generalized for the system of n equations. Equations (3.50a) – (3.53b) can be written in the general forms as follows,

$$\ell_{il} = a_{il} \quad i = 1, 2, \dots, n \quad (3.55a)$$

$$u_{1j} = \frac{a_{1j}}{\ell_{11}} \quad j = 2, 3, \dots, n \quad (3.55b)$$

and for $j = 2, 3, \dots, n-1$;

$$\ell_{ij} = a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \quad i = j, j+1, \dots, n \quad (3.55c)$$

$$u_{jk} = \frac{a_{jk} - \sum_{i=1}^{j-1} \ell_{ji} u_{ik}}{\ell_{jj}} \quad k = j+1, j+2, \dots, n \quad (3.55d)$$

with,

$$\ell_{mn} = a_{mn} - \sum_{k=1}^{n-1} \ell_{nk} u_{kn} \quad (3.55e)$$

After the matrices $[L]$ and $[U]$ are obtained, the vectors $\{Y\}$ and $\{X\}$ can be determined from the formulas,

$$\begin{aligned} y_1 &= \frac{b_1}{\ell_{11}} \\ y_i &= \frac{b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j}{\ell_{ii}} \quad i = 2, 3, \dots, n \end{aligned} \quad (3.56a)$$

and

$$\begin{aligned} x_n &= y_n \\ x_i &= y_i - \sum_{j=i+1}^n u_{ij} x_j \quad i = n-1, n-2, \dots, 1 \end{aligned} \quad (3.56b)$$

The procedures above are used to develop a corresponding computer program as shown in Fig. 3.5. Table 3.3 shows example of the input data file of Eq. (3.10) and the computed solutions generated from the program.

```
% Program LUDCom
% A program for solving a set of linear
% simultaneous equations [A]{X} = {B} by
% using the LU decomposition method.
% Read input data file containing number
% of eqs with matrix [A] and vector {B}:
fid = fopen('input.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [n+1 n]); a = a.';
b = squeeze(a(:,n+1));
a = squeeze(a(:,1:n));
fclose(fid);
x = LU(n, a, b);
% Print out the computed solution of {X}:
fid = fopen('output.dat', 'w');
fprintf(fid, ...
'\n Equation No.      Solution x \n');
fprintf(
...
'\n Equation No.      Solution x \n');
for i = 1:n
    fprintf(fid, ' %8d %22.6e\n', i, x(i));
    fprintf(    ' %8d %22.6e\n', i, x(i));
end
fclose(fid);



---


function x = LU(n, a, b)
% Perform decomposition [A] = [L][U]:
al = zeros(n,n); au = zeros(n,n);
y = zeros(n,1);
for i = 1:n
    al(i,1) = a(i,1);
end
for j = 2:n
    au(1,j) = a(1,j)/al(1,1);
end
for j = 2:n-1
    for i = j+1:n
        sum = 0;
        for k = 1:j-1
            sum = sum + au(k,j)*y(k);
        end
        y(i) = (b(i) - sum)/al(i,i);
    end
    % Backward pass to solve [U]{X} = {Y}
    x(n) = y(n);
    for i = n-1:-1:1
        sum = 0;
        for j = i+1:n
            sum = sum + au(i,j)*x(j);
        end
        x(i) = y(i) - sum;
    end
end
```

Figure 3.5 Computer program for solving system of equations by the LU decomposition method.

Table 3.3 Example of input data file for Eq. (3.10) for the LU decomposition computer program as shown in Fig. 3.5 and its computed solutions.

<u>Input data for Eq. (3.10)</u>	<u>Computational results</u>	
3	Equation No.	Solution x
4. -4. 0. 400.	1	.450000E+03
-1. 4. -2. 400.	2	.350000E+03
0. -2. 4. 400.	3	.275000E+03

The number of numerical operations required by the LU decomposition method to solve a system of equations is close to that needed by the Gauss elimination method. However, the LU decomposition method requires more computer memory to store the two matrices $[L]$ and $[U]$ as compared to the Gauss elimination method. The required computer memory may be reduced by developing a more complex computer program.

3.10 MATLAB Function for LU Decomposition

MATLAB uses the built-in function `lu` to decompose the matrix $[A]$ into the two matrices $[L]$ and $[U]$. The syntax for this function is,

$$[L, U] = \text{lu}(A)$$

where L , U , and A represent the matrices $[L]$, $[U]$ and $[A]$, respectively.

Example 3.14 Solve the following system of equations by using MATLAB with the LU decomposition method,

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

The two matrices $[A]$ and $\{B\}$ are first defined by using the commands,

```
>> A = [4 -4 0; -1 4 -2; 0 -2 4];
>> B = [400; 400; 400];
```

Then, the function `lu` is employed to decompose the matrix $[A]$ into the two matrices $[L]$ and $[U]$,

```
>> [L,U] = lu(A)
L =
    1.0000      0      0
   -0.2500    1.0000      0
    0     -0.6667    1.0000
U =
    4.0000    -4.0000      0
    0      3.0000   -2.0000
    0          0    2.6667
```

The computed matrices $[L]$ and $[U]$ can be verified by multiplying them together. The result must be equal to the original matrix $[A]$, i.e.,

```
>> L*U
ans =
    4     -4      0
   -1      4     -2
    0     -2      4
```

To solve the system of equations by the LU decomposition method, the left division is used in the two following steps,

```
>> Y = L\B
Y =
400.0000
500.0000
733.3333
>> x = U\Y
x =
450.0000
350.0000
275.0000
```

The above example shows that the roots of a system of equations can be obtained easily by using the LU decomposition method in MATLAB. Understanding the method procedure, however, is required prior to using the function `lu` built in MATLAB.

3.11 Cholesky Decomposition Method

All methods explained earlier in this chapter can be used to solve the system of equations $[A]\{X\} = \{B\}$ where the square matrix $[A]$ may be a symmetric or an asymmetric matrix. For many engineering problems, the matrix $[A]$ generated from the finite difference or finite element technique is normally a symmetric matrix. If the matrix $[A]$ is a symmetric matrix, the LU decomposition method can be modified so that its solving procedure is reduced. Such modified method is known as the Cholesky decomposition method. The method is often used for solving linear structural problems for which their matrices $[A]$ are symmetric. The Cholesky decomposition method starts from the system of equations,

$$[A]\{X\} = \{B\} \quad (3.4)$$

by decomposing the symmetric matrix $[A]$ into the form,

$$[A] = [L][L]^T \quad (3.57)$$

where $[L]$ is the matrix that contains all zero coefficients above its diagonal line. For example, if the system of Eq. (3.4) consists of 3 equations, the matrix $[A]$ is decomposed into the form,

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}}_{[A]} = \underbrace{\begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix}}_{[L]} \underbrace{\begin{bmatrix} \ell_{11} & \ell_{21} & \ell_{31} \\ 0 & \ell_{22} & \ell_{32} \\ 0 & 0 & \ell_{33} \end{bmatrix}}_{[L]^T} \quad (3.58)$$

The coefficients in the matrix $[L]$ can be determined as follows,

$$\begin{aligned}\ell_{11}^2 &= a_{11} \rightarrow \ell_{11} = \sqrt{a_{11}} \\ \ell_{11}\ell_{21} &= a_{12} \rightarrow \ell_{21} = a_{12}/\ell_{11} \\ \ell_{11}\ell_{31} &= a_{13} \rightarrow \ell_{31} = a_{13}/\ell_{11} \\ \ell_{21}^2 + \ell_{22}^2 &= a_{22} \rightarrow \ell_{22} = \sqrt{a_{22} - \ell_{21}^2} \\ \ell_{21}\ell_{31} + \ell_{22}\ell_{32} &= a_{23} \rightarrow \ell_{32} = (a_{23} - \ell_{21}\ell_{31})/\ell_{22} \\ \ell_{31}^2 + \ell_{32}^2 + \ell_{33}^2 &= a_{33} \rightarrow \ell_{33} = \sqrt{a_{33} - \ell_{31}^2 - \ell_{32}^2}\end{aligned}$$

By substituting Eq. (3.58) into Eq. (3.4),

$$\underbrace{\begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} \ell_{11} & \ell_{21} & \ell_{31} \\ 0 & \ell_{22} & \ell_{32} \\ 0 & 0 & \ell_{33} \end{bmatrix} \underbrace{\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix}}_{\{X\}}}_{\{Y\}} = \underbrace{\begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}}_{\{B\}} \quad (3.59)$$

The vector $\{Y\}$ is determined by using forward substitution,

$$\begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \quad (3.60)$$

Then, the vector $\{X\}$ is determined by using back substitution,

$$\begin{bmatrix} \ell_{11} & \ell_{21} & \ell_{31} \\ 0 & \ell_{22} & \ell_{32} \\ 0 & 0 & \ell_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix} \quad (3.61)$$

The procedures for determining the coefficients in the matrix $[L]$ as shown in the above example can be generalized for the system of n equations as follows.

For the first row of matrix $[L]$,

$$\ell_{11} = \sqrt{a_{11}} \quad (3.62a)$$

For the next k^{th} row, where $k = 2, 3, \dots, n$,

$$\ell_{ki} = \frac{a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} \ell_{kj}}{\ell_{ii}} \quad i = 1, 2, \dots, k-1 \quad (3.62b)$$

with

$$\ell_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} \ell_{kj}^2} \quad (3.62c)$$

After the matrix $[L]$ is obtained, the vectors $\{Y\}$ and $\{X\}$ can be determined, respectively, as

$$\begin{aligned} y_1 &= \frac{b_1}{\ell_{11}} \\ y_i &= \frac{b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j}{\ell_{ii}} \quad i = 2, 3, \dots, n \end{aligned} \quad (3.63)$$

and

$$\begin{aligned} x_n &= \frac{y_n}{\ell_{nn}} \\ x_i &= \frac{y_i - \sum_{j=i+1}^n \ell_{ji} x_j}{\ell_{ii}} \quad i = n-1, n-2, \dots, 1 \end{aligned} \quad (3.64)$$

Example 3.15 Develop a computer program to solve the system of n equations in the form of $[A]\{X\} = \{B\}$ where the matrix $[A]$ is symmetric. Verify the computer program by solving the following system of equations,

$$\begin{bmatrix} 4 & 3 & 1 \\ 3 & 5 & 2 \\ 1 & 2 & 6 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 3,125 \\ 3,650 \\ 2,800 \end{Bmatrix} \quad (3.65)$$

where the exact solutions are $x_1 = 450$, $x_2 = 350$ and $x_3 = 275$.

A computer program for solving the system of n equations in the form of $[A]\{X\} = \{B\}$ by using the Cholesky decomposition method can be developed as shown in Fig. 3.6. The program is verified by solving Eq. (3.65) with the computed solutions as shown in Table 3.4.

Table 3.4 Input data file of Eq. (3.65) for the computer program as shown in Fig. 3.6 and the computed solutions from the Cholesky decomposition method.

Input data for Eq. (3.65)

3			
4.	3.	1.	3125.
3.	5.	2.	3650.
1.	2.	6.	2800.

Computed solutions

Equation No.	Solution x
1	.450000E+03
2	.350000E+03
3	.275000E+03

```

% Program Cholesky
% A program for solving a set of linear
% simultaneous equations [A]{X} = {B} by
% using Cholesky decomposition method.
% Read input data file containing number
% of eqs with matrix [A] and vector {B}:
fid = fopen('inputcholes.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [n+1 n]); a = a.';
b = squeeze(a(:, n+1));
a = squeeze(a(:, 1:n));
fclose(fid);
x = Choles(n, a, b);
% Print out the computed solution of {X}:
fid = fopen('output.dat', 'w');
fprintf(fid, ...
'\n Equation No.      Solution x \n');
fprintf( ...
'\n Equation No.      Solution x \n');
for i = 1:n
    fprintf(fid, '%8d %22.6e\n', i, x(i));
    fprintf( ' %8d %22.6e\n', i, x(i));
end
fclose(fid);

function x = Choles(n, a, b)
% Perform decomposition [A] = [L] [U]:
al = zeros(n,n); y = zeros(n,1);
al(1,1) = sqrt(a(1,1));
for k = 2:n
    for i = 1:k-1
        sum = 0.;

    if(i ~= 1)
        for j = 1:i-1
            sum = sum + al(i,j)*al(k,j);
        end
        al(k,i) = (a(k,i) - sum)/al(i,i);
    end
    sum = 0.;
    for j = 1:k-1
        sum = sum + al(k,j)*al(k,j);
    end
    al(k,k) = sqrt(a(k,k) - sum);
end
% Perform forward pass:
y(1) = b(1)/al(1,1);
for i = 2:n
    sum = 0.;
    for j = 1:i-1
        sum = sum + al(i,j)*y(j);
    end
    y(i) = (b(i) - sum)/al(i,i);
end
% Perform backward pass:
x(n) = y(n)/al(n,n);
for i = n-1:-1:1
    sum = 0.;
    for j = i+1:n
        sum = sum + al(j,i)*x(j);
    end
    x(i) = (y(i) - sum)/al(i,i);
end

```

Figure 3.6 Computer program for solving a system of n equations by using the Cholesky decomposition method.

3.12 MATLAB Function for Cholesky Decomposition

MATLAB uses the built-in function `chol` to decompose the matrix $[A]$. The syntax for this function is

$$U = \text{chol}(A)$$

where U and A represents the matrices $[L]^T$ and $[A]$ as shown in Eq. (3.57), respectively.

Example 3.16 Solve the following system of equations by using the MATLAB function `chol` for the Cholesky decomposition method.

$$\begin{bmatrix} 4 & 3 & 1 \\ 3 & 5 & 2 \\ 1 & 2 & 6 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 3,125 \\ 3,650 \\ 2,800 \end{Bmatrix}$$

The matrices $[A]$ and $\{B\}$ are first defined by the commands,

```

>> A = [4 3 1; 3 5 2; 1 2 6];
>> B = [3125; 3650; 2800];

```

The function `chol` is then used to decompose matrix $[A]$,

```
>> U = chol(A)
U =
    2.0000    1.5000    0.5000
        0    1.6583    0.7538
        0         0    2.2764
```

The result from decomposing can be verified by,

```
>> U' *U
ans =
    4      3      1
    3      5      2
    1      2      6
```

Finally, the left division command is applied to obtain the solutions as follows,

```
>> Y = U' \B
Y =
1.0e+003 *
    1.5625
    0.7877
    0.6260
>> x = U\Y
x =
    450.0000
    350.0000
    275.0000
```

The computed solutions above are the exact solutions of the problem.

3.13 Jacobi Iteration Method

All methods for solving a system of equations explained so far are called the direct technique because the solutions are obtained directly from certain computational procedures. The technique, however, requires substantial computational time and memory when solving a large set of equations. Another group of methods, called the iterative technique, are sometimes used to solve a set of equations that arises from some types of engineering problems.

The simplest iterative technique is the Jacobi iteration method. The method can be understood easily by considering the following system of equations,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \quad (3.66)$$

The basic idea behind this method is to rewrite each equation so that its unknown appears on the left-hand side of that equation,

$$x_1 = \frac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \quad (3.67a)$$

$$x_2 = \frac{b_2 - a_{21}x_1 - a_{23}x_3}{a_{22}} \quad (3.67b)$$

$$x_3 = \frac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \quad (3.67c)$$

The computational procedure starts from using a set of guess values x_1, x_2, x_3 on the right-hand side of Eq. (3.67a-c) in order to compute the new values x_1, x_2, x_3 . The process is repeated until the updated values of x_1, x_2, x_3 converge to the solutions within the specified tolerance ε ,

$$\left| \frac{x_i^{k+1} - x_i^k}{x_i^{k+1}} \right| \times 100\% < \varepsilon \quad (3.68)$$

where the subscript i is the equation number and the superscript k is the k^{th} iteration.

The system of equation (3.10) is solved again but by using the Jacobi iteration method as shown in the example below.

Example 3.17 Use the Jacobi iteration method to solve Eq. (3.10),

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

The three equations in Eq. (3.10) are,

$$4x_1 - 4x_2 = 400 \quad (3.69a)$$

$$-x_1 + 4x_2 - 2x_3 = 400 \quad (3.69b)$$

$$-2x_2 + 4x_3 = 400 \quad (3.69c)$$

These three equations are written such that their unknowns appear on the left-hand-side of the equations,

$$x_1 = 100 + x_2 \quad (3.70a)$$

$$x_2 = 100 + \frac{1}{4}x_1 + \frac{1}{2}x_3 \quad (3.70b)$$

$$x_3 = 100 + \frac{1}{2}x_2 \quad (3.70c)$$

Then, the iteration numbers are included to indicate the old and new values of the solutions,

$$x_1^{k+1} = 100 + x_2^k \quad (3.71a)$$

$$x_2^{k+1} = 100 + \frac{1}{4}x_1^k + \frac{1}{2}x_3^k \quad (3.71b)$$

$$x_3^{k+1} = 100 + \frac{1}{2}x_2^k \quad (3.71c)$$

where the superscript k represents the k^{th} iteration.

For example, if the initial guess values are $x_1 = x_2 = x_3 = 100$, then the new values of x_1, x_2, x_3 in Eq. (3.71a-c) are

$$x_1 = 100 + 100 = 200 \quad (3.72a)$$

$$x_2 = 100 + \frac{1}{4}(100) + \frac{1}{2}(100) = 175 \quad (3.72b)$$

$$x_3 = 100 + \frac{1}{2}(100) = 150 \quad (3.72c)$$

The differences between the old and new values are determined according to Eq. (3.68) as,

$$\text{Difference of } x_1 = \left| \frac{100 - 200}{200} \right| \times 100\% = 50.00\% \quad (3.73a)$$

$$\text{Difference of } x_2 = \left| \frac{100 - 175}{175} \right| \times 100\% = 42.86\% \quad (3.73b)$$

$$\text{Difference of } x_3 = \left| \frac{100 - 150}{150} \right| \times 100\% = 33.33\% \quad (3.73c)$$

The process is repeated until the differences of the old and new values of x_1, x_2, x_3 are less than a specified tolerance ε , such as $\varepsilon = 0.05\%$.

Figure 3.7 shows a computer program for solving the system of equations in this example. The initial values of x_1, x_2, x_3 used in the program are all 100 and the specified tolerance is $\varepsilon = 0.05\%$. The computed solutions at different iterations are shown in Table 3.5. The table shows that the solutions converge to the exact solution within 19 iterations. Figure 3.7 and Table 3.5 demonstrate that a computer program using the Jacobi iteration method can be developed so that the solutions from a set of equations are obtained easily.

```
% Program Jacobi
% Example of Jacobi iteration method.
tol = 0.05;
for i = 1:3
    xold(i) = 100.;
end
fprintf('%3s %4s %9s %9s\n', ...
    'Iteration No.', 'x1', 'x2', 'x3');
for iter = 1:500
    xnew(1)=100.+xold(2);
    xnew(2)=100.+0.25*xold(1) + 0.5*xold(3);
    xnew(3)=100.+0.50*xold(2);
    fprintf('%8d%10.0f%10.0f%10.0f\n', ...
        iter,xold(1),xold(2),xold(3));
    iflag = 0;
    for i = 1:3
        if(xnew(i) ~= xold(i))
            iflag = 1;
        end
    end
    if(iflag == 0)
        break
    end
    for i = 1:3
        xold(i) = xnew(i);
    end
end
```

Figure 3.7 Computer program for solving the system of Eq. (3.10) by the Jacobi iteration method.

Table 3.5 The computed solutions of Eq. (3.10) at each iteration by using the Jacobi iteration method.

Iteration No.	x1	x2	x3
1	100.	100.	100.
2	200.	175.	150.
3	275.	225.	188.
4	325.	263.	213.
5	363.	288.	231.
6	388.	306.	244.
7	406.	319.	253.
8	419.	328.	259.
9	428.	334.	264.
10	434.	339.	267.
11	439.	342.	270.
12	442.	345.	271.
13	445.	346.	272.
14	446.	347.	273.
15	447.	348.	274.
16	448.	349.	274.
17	449.	349.	274.
18	449.	349.	275.
19	449.	350.	275.

3.14 Gauss-Seidel Iteration Method

The Gauss-Seidel method is similar to the Jacobi method explained in the preceding section but can increase the solution convergence rate. The Gauss-Seidel method reduces the number of iterations by using the updated values of x_1, x_2, x_3 right after they were computed. The example below shows the Gauss-Seidel method for solving the same set of Eq. (3.10).

Example 3.18 Use the Gauss-Seidel iteration method to solve the set of equations,

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

The same equations as shown in Example 3.17 can be used for the Gauss-Seidel iteration method except Eqs. (3.71b-c). The newly computed values of x_1 and x_2 are used immediately in the calculation for the new x_2 and x_3 in Eqs. (3.74b-c), respectively.

$$x_1^{k+1} = 100 + x_2^k \quad (3.74a)$$

$$x_2^{k+1} = 100 + \frac{1}{4}x_1^{k+1} + \frac{1}{2}x_3^k \quad (3.74b)$$

$$x_3^{k+1} = 100 + \frac{1}{2}x_2^{k+1} \quad (3.74c)$$

For example, if the initial guess values of $x_1 = x_2 = x_3 = 100$, then the new values of x_1 , x_2 , and x_3 can be determined from Eq. (3.74a-c) as,

$$x_1 = 100 + 100 = 200 \quad (3.75a)$$

$$x_2 = 100 + \frac{1}{4}(200) + \frac{1}{2}(100) = 200 \quad (3.75b)$$

$$x_3 = 100 + \frac{1}{2}(200) = 200 \quad (3.75c)$$

The process is repeated until the differences between the old and new values of x_1 , x_2 and x_3 are less than the specified tolerance ε .

Figure 3.8 shows a computer program developed for this example. The initial values of x_1 , x_2 , x_3 are set to 100 with the specified tolerance of $\varepsilon = 0.05\%$. The computed solutions obtained from this program are shown in Table 3.6. The table shows that the variables x_1 , x_2 , x_3 converge to the final solutions within 12 iterations as compared to 19 iterations by the Jacobi method.

```
% Program GSeidel
% Example of Gauss-Seidel iteration method.
tol = 0.05;
for i = 1:3
    xold(i) = 100.;
end
fprintf('%3s %4s %9s %9s\n', ...
    'Iteration No.', 'x1', 'x2', 'x3');
for iter = 1:500
    xnew(1)=100.+xold(2);
    xnew(2)=100.+0.25*xnew(1) + 0.5*xold(3);
    xnew(3)=100.+0.50*xnew(2);
    fprintf('%8d%10.0f%10.0f%10.0f\n', ...
        iter,xold(1),xold(2),xold(3));
    iflag = 0;
    for i = 1:3
        xold(i) = xnew(i);
    end
    if iflag == 0
        break
    end
end
```

Figure 3.8 Computer program for solving the system of Eq. (3.10) by the Gauss-Seidel iteration method.

Table 3.6 The computed solutions of Eq. (3.10) by using Gauss-Seidel iteration method.

Iteration No.	x1	x2	x3
1	100.	100.	100.
2	200.	200.	200.
3	300.	275.	238.
4	375.	313.	256.
5	413.	331.	266.
6	431.	341.	270.
7	441.	345.	273.
8	445.	348.	274.
9	448.	349.	274.
10	449.	349.	275.
11	449.	350.	275.
12	450.	350.	275.

3.15 Successive Over-relaxation Method

The successive over-relaxation method is modified from Gauss-Seidel method in order to further increase the solution convergence rate. The new solutions are obtained by weighting between the newly computed and the old values as,

$$x_i^{k+1} = \omega x_i^{k+1*} + (1-\omega)x_i^k \quad (3.76)$$

where the superscript k represent the k^{th} iteration, x_i^{k+1*} is the newly computed value as explained in the Gauss-Seidel method and ω is the weighting factor which has a value between 0 and 2. If $\omega=1$, the method becomes the Gauss-Seidel method. If the value of ω is between 1 and 2, the computed solutions are weighted by the new values more than the old values, which is called over-relaxation. If ω is between 0 and 1, the method is called under-relaxation that may help a nearly diverged solution to converge. In general, the value of ω should be selected between 1 and 2 to increase the convergence rate. It is noted that the appropriate value of ω varies depending on the problems.

To understand the successive over-relaxation method clearly, Eq. (3.10) is solved again as shown in the following example.

Example 3.19 Use the successive over-relaxation method to solve Eq. (3.10),

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

From the Gauss-Seidel method, the new values of x_1, x_2, x_3 at the $(k+1)^{th}$ iteration are

$$x_1^{k+1} = 100 + x_2^k \quad (3.74a)$$

$$x_2^{k+1} = 100 + \frac{1}{4}x_1^{k+1} + \frac{1}{2}x_3^k \quad (3.74b)$$

$$x_3^{k+1} = 100 + \frac{1}{2}x_2^{k+1} \quad (3.74c)$$

With the successive over-relaxation, the new values of x_1, x_2, x_3 are

$$x_1^{k+1} = \omega(100 + x_2^k) + (1-\omega)x_1^k \quad (3.77a)$$

$$x_2^{k+1} = \omega\left(100 + \frac{1}{4}x_1^{k+1} + \frac{1}{2}x_3^k\right) + (1-\omega)x_2^k \quad (3.77b)$$

$$x_3^{k+1} = \omega\left(100 + \frac{1}{2}x_2^{k+1}\right) + (1-\omega)x_3^k \quad (3.77c)$$

Figure 3.9 shows a computer program developed with the initial value x_1, x_2, x_3 of 100, the weighting factor $\omega=1.2$ and the specified tolerance $\varepsilon=0.05\%$. The computed solutions obtained from the computer program are shown in Table 3.7. The results show that the converged solutions are obtained within 6 iterations as compared to 12 and 19 iterations by the Gauss-Seidel and Jacobi methods, respectively.

```
% Program SOR
% Example of the Successive Over
% Relaxation method.
w = 1.2; tol = 0.05;
for i = 1:3
    xold(i) = 100.;
end
fprintf('%3s %4s %9s %9s\n', ...
    'Iteration No.', 'x1', 'x2', 'x3');
for iter = 1:500
    xnew(1) = 100.+xold(2);
    xnew(1) = w*xnew(1)+(1-w)*xold(1);
    xnew(2) = 100.+0.25*xnew(1)+0.5*xold(3);
    xnew(2) = w*xnew(2)+(1-w)*xold(2);
    xnew(3) = 100.+0.50*xnew(2);
    xnew(3) = w*xnew(3)+(1-w)*xold(3);
    fprintf('%8d%10.0f%10.0f%10.0f\n', ...
        iter,xold(1),xold(2),xold(3));
    iflag = 0;
    for i = 1:3
        eps = abs((xnew(i) - xold(i))*100. ...
            /xnew(i));
        if(eps >= tol)
            iflag = 1;
        end
    end
    if iflag == 0
        break
    end
    for i = 1:3
        xold(i) = xnew(i);
    end
end
```

Figure 3.9 Computer program for solving the system of Eq. (3.10) by the successive over-relaxation method.

Table 3.7 The computed solutions of Eq. (3.10) at each iteration by using the successive over-relaxation method.

Iteraion No.	x1	x2	x3
1	100.	100.	100.
2	220.	226.	236.
3	347.	320.	265.
4	435.	345.	274.
5	448.	350.	275.
6	450.	350.	275.

3.16 Conjugate Gradient Method

One of the most popular methods for solving a system of equations based on the iterative technique is the conjugate gradient method. The method is used for solving the system of equations in the form

$$[A] \{X\} = \{B\} \quad (3.4)$$

where the matrix $[A]$ is a symmetric and positive definite matrix. Definitions of a positive definite matrix will be explained later. The method provides a converged solution within n iterations if the system of equations consists of n equations. In addition, the method performs effectively when the matrix $[A]$ is a sparse matrix.

In order to understand the concept of the method, an example for the system of two equations ($n = 2$) is considered,

$$\begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 4 \\ -3 \end{Bmatrix} \quad (3.78)$$

where $[A]$ is a symmetric matrix with its dimensions of (2×2) . Equation (3.78) are written in the form of two algebraic equations as,

$$2x_1 + x_2 = 4 \quad (3.79)$$

$$x_1 + 3x_2 = -3 \quad (3.80)$$

These two equations can be plotted and represented by the two straight lines as shown in Fig. 3.10. The two lines intersect at the point where $x_1 = 3$ and $x_2 = -2$ which are the solutions of the system of equations.

$$\{X\} = \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix} = \begin{Bmatrix} 3 \\ -2 \end{Bmatrix} \quad (3.81)$$

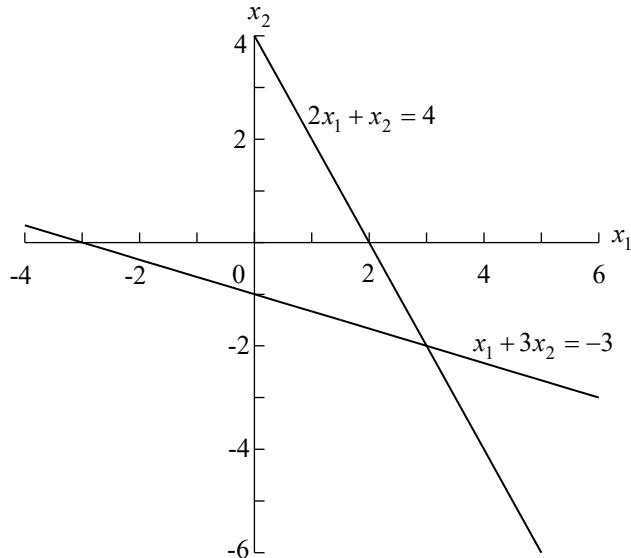


Figure 3.10 Two straight lines represented by Eqs. (3.79 - 3.80) and the intersection point at $x_1 = 3$ and $x_2 = -2$.

The procedures to find the solutions of Eq. (3.78) by using the conjugate gradient method start from the quadratic function in the form,

$$f(x_1, x_2) = \frac{1}{2} \begin{bmatrix} X \\ 1 \times 2 \end{bmatrix} \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} X \\ 2 \times 2 \end{bmatrix} - \begin{bmatrix} B \\ 2 \times 1 \end{bmatrix} \begin{bmatrix} X \\ 2 \times 1 \end{bmatrix} \quad (3.82)$$

or

$$f(x_1, x_2) = \frac{1}{2} \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 4 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (3.83)$$

The function $f(x_1, x_2)$ in Eq. (3.83) is plotted by using contour lines as shown in Fig. 3.11. The figure shows that the quadratic function $f(x_1, x_2)$ is minimum at the location where x_1 and x_2 are the solutions of the system of equations.

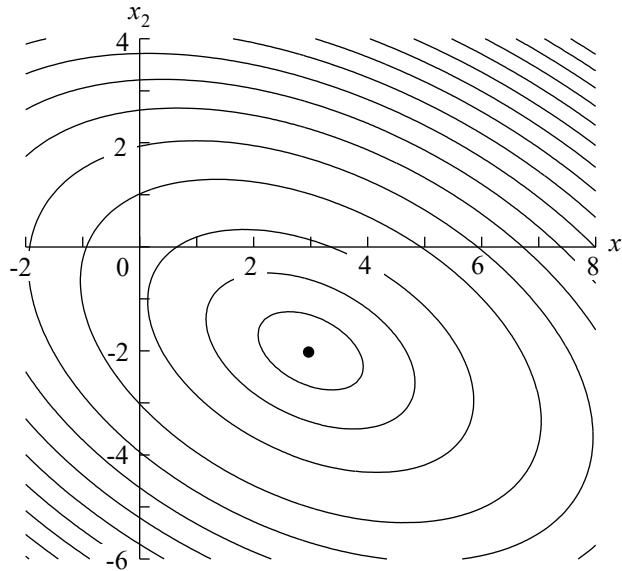


Figure 3.11 Contour plot of the quadratic function, $f(x_1, x_2)$.

If the function $f(x_1, x_2)$ is plotted in three dimensions, the shape of the function is similar to a blunt cone as shown in Fig. 3.12. The location where $x_1 = 3$ and $x_2 = -2$ is at the bottom of the blunt cone.

At the bottom location of the blunt cone with the solutions of x_1 and x_2 , the gradient of the quadratic function is zero. Since Eq. (3.83) is,

$$f(x_1, x_2) = \frac{1}{2} (2x_1^2 + 2x_1x_2 + 3x_2^2) - 4x_1 + 3x_2 \quad (3.84)$$

the first derivatives with respect to x_1 and x_2 representing zero gradient at this location are,

$$\frac{\partial f}{\partial x_1} = 2x_1 + x_2 - 4 = 0 \quad (3.85a)$$

$$\frac{\partial f}{\partial x_2} = x_1 + 3x_2 + 3 = 0 \quad (3.85b)$$

The results are the original system of Eq. (3.78). The process shows that the solutions of system of equations can be determined from the condition of zero derivatives of the quadratic function.

To find the solutions of the system of equations by the conjugate gradient method, the quadratic function must vary as a blunt cup shape similar to that shown in Fig. (3.12). Such blunt cup shape occurs when the matrix $[A]$ is positive definite, i.e.,

$$[X][A]\{X\} > 0 \quad (3.86)$$

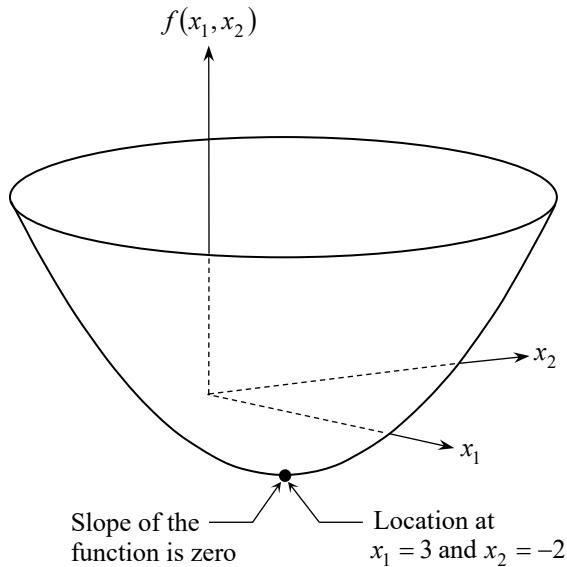


Figure 3.12 Shape of the quadratic function leading to the concept of the conjugate gradient method.

where $\{X\}$ is a non-zero vector. Verifying that a matrix $[A]$ is a positive definite by using Eq. (3.86) is cumbersome if its dimensions are large. An alternative way for verifying a positive definite matrix $[A]$ is to observe its diagonal coefficients. If they are all positive, the matrix $[A]$ is likely a positive definite. Another way is to determine its determinant as follows. For the matrix $[A]$ with its dimensions of $(n \times n)$, the following determinants,

$$D_1 = |a_{11}| \quad (3.87a)$$

$$D_2 = \begin{vmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{vmatrix} \quad (3.87b)$$

$$D_3 = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{vmatrix} \quad (3.87c)$$

up to

$$D_n = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{vmatrix} \quad (3.87n)$$

must be greater than zero. For example, the matrix $[A]$ in Eq. (3.78) is positive definite because its $D_1 = 2$ and $D_2 = 5$.

To illustrate the procedure of the conjugate gradient method for solving the system of n equations in Eq. (3.4), the quadratic function

$$f(x_1, x_2, \dots, x_n) = \frac{1}{2} \underbrace{\begin{bmatrix} X \\ (1 \times n) \end{bmatrix}}_{(n \times n)} \underbrace{\begin{bmatrix} A \\ (n \times n) \end{bmatrix}}_{(n \times 1)} \underbrace{\begin{bmatrix} X \\ (1 \times n) \end{bmatrix}}_{(n \times 1)} - \underbrace{\begin{bmatrix} B \\ (1 \times n) \end{bmatrix}}_{(n \times 1)} \underbrace{\begin{bmatrix} X \\ (1 \times n) \end{bmatrix}}_{(n \times 1)} \quad (3.88)$$

is used. Their first derivatives with respect to x_1, x_2, \dots, x_n are determined and set to zero as,

$$\frac{\partial f}{\partial \{X\}} = [A]\{X\} - \{B\} = 0 \quad (3.89)$$

The result is identical Eq. (3.4) if the matrix $[A]$ in Eq. (3.89) is positive definite.

Because the conjugate gradient method is an iterative technique, a set of initial guess values in the vector $\{X\}$ is needed. Such set of values may represent a point on the surface of the blunt cone. The idea of the conjugate gradient method is to move from that point to the bottom location of the blunt cone representing the solutions of the system of equations. For a system of n equations, the shape of function f is a complex shape in hyperspace (x_1, x_2, \dots, x_n) which can not be displayed in three dimensions. The iterative process updates the vector $\{X\}$ by using its old solutions to determine the new solutions at the $(k+1)^{th}$ iteration as follows,

$$\{X\}^{k+1} = \{X\}^k + \lambda_k \{D\}^k \quad (3.90)$$

where $\{D\}^k$ is called the *search direction vector*. It is the vector that searches for the direction to the bottom location of the blunt cone. The scalar quantity λ_k represents the step size of the vector $\{D\}^k$ which changes at each iteration.

The value of λ_k for the k^{th} iteration can be determined by substituting Eq. (3.90) into Eq. (3.88), taking the first derivative with respect to λ_k and setting them to zero as,

$$\frac{\partial f}{\partial \lambda_k} = \lfloor D \rfloor^k [A] \{D\}^k \lambda_k + \lfloor D \rfloor^k \{R\}^k = 0 \quad (3.91)$$

where

$$\{R\}^k = [A]\{X\}^k - \{B\} \quad (3.92)$$

is the residual vector. The vector approaches zero as the computed solutions converge to the exact solutions. Equation (3.91) can be written for determining the value λ_k as,

$$\lambda_k = - \frac{\lfloor D \rfloor^k \{R\}^k}{\lfloor D \rfloor^k [A] \{D\}^k} \quad (3.93)$$

It should be again noted that Eq. (3.93) is valid if and only if the matrix $[A]$ is positive definite.

During the iteration process, if the vector $\{X\}^k$ at the k^{th} iteration is not the final solution, then the residual vector $\{R\}^k$ from Eqs. (3.89) and (3.92) is not zero,

$$\frac{\partial f}{\partial \{X\}^k} = [A]\{X\}^k - \{B\} = \{R\}^k \quad (3.94)$$

The residual vector $\{R\}^k$ represents the gradient of function f for the solutions in the vector $\{X\}^k$. Thus, if the vector $\{X\}^k$ contains the solutions of the system of equations at the bottom of the blunt cone, the residual vector $\{R\}^k$ representing the gradient of function must be zero.

At the first iteration ($k = 0$), the computational process starts from the initial guess vector $\{X\}^0$ and the search direction vector $\{D\}^0$. The search direction vector $\{D\}^0$ is first assigned in the opposite direction of the residual vector $\{R\}^0$ as,

$$\{D\}^0 = -\{R\}^0 \quad (3.95)$$

A new search direction vector $\{D\}$ is then determined from

$$\{D\}^{k+1} = -\{R\}^{k+1} + \alpha_k \{D\}^k \quad (3.96)$$

where α_k is the value that makes the vector $\{D\}^{k+1}$ to be $[A]$ -conjugated with the vector $\{D\}^k$, i.e.,

$$\{D\}^{k+1} [A] \{D\}^k = 0 \quad (3.97)$$

By substituting Eq. (3.96) into Eq. (3.97) and arranging the equation for determining the value of α_k ,

$$\alpha_k = \frac{[\{R\}]^{k+1} [A] \{D\}^k}{[\{D\}]^k [A] \{D\}^k} \quad (3.98)$$

It is noted that the word “conjugate” in the conjugate gradient method came from the requirement that if a vector $\{u\}$ is $[A]$ -conjugated with another vector $\{v\}$, then

$$[u] [A] \{v\} = 0 \quad (3.99)$$

where the matrix $[A]$ must be symmetric and positive definite.

In conclusion, the procedures of the conjugate gradient method for solving the system of n equations,

$$\begin{matrix} [A] \{X\} \\ (n \times n) (n \times 1) \end{matrix} = \begin{matrix} \{B\} \\ (n \times 1) \end{matrix} \quad (3.4)$$

where the matrix $[A]$ is symmetric and positive definite are as follows,

1. Assume initial vector $\{X\}^0$ and calculate the residual vector,

$$\{R\}^0 = [A] \{X\}^0 - \{B\}$$

2. Assign the search direction vector,

$$\{D\}^0 = -\{R\}^0$$

3. Start the iteration process for $k = 0, 1, 2, \dots, n$ to determine,

$$\lambda_k = -\frac{\lfloor D \rfloor^k \{R\}^k}{\lfloor D \rfloor^k [A] \{D\}^k}$$

$$\{X\}^{k+1} = \{X\}^k + \lambda_k \{D\}^k$$

$$\{R\}^{k+1} = [A]\{X\}^{k+1} - \{B\}$$

In each iteration, determine and examine the error using $Error = \sqrt{\lfloor R \rfloor^{k+1} \{R\}^{k+1}} < \varepsilon$, where ε represents the acceptable error. If the condition is true, stop the iteration process. If not, the iteration process continues by determining,

$$\alpha_k = \frac{\lfloor R \rfloor^{k+1} [A] \{D\}^k}{\lfloor D \rfloor^k [A] \{D\}^k}$$

$$\{D\}^{k+1} = -\{R\}^{k+1} + \alpha_k \{D\}^k$$

and go back to step 3 for the new k^{th} iteration.

The above procedures are developed as a computer program as shown in Fig. 3.13 with detailed computations as shown in the example below.

Example 3.20 Use the conjugate gradient method to solve the system of equations,

$$\begin{bmatrix} 4 & -4 & 0 \\ -4 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ -950 \\ 400 \end{Bmatrix} \quad (3.100)$$

Compare the computed solutions with the exact solutions of $x_1 = 450$, $x_2 = 350$ and $x_3 = 275$.

Solution Assume the initial guess values in the vector $\{x\}^0 = \begin{Bmatrix} 100 \\ 100 \\ 100 \end{Bmatrix}$, then

$$\{R\}^0 = [A]\{X\}^0 - \{B\} = \begin{bmatrix} 4 & -4 & 0 \\ -4 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} 100 \\ 100 \\ 100 \end{Bmatrix} - \begin{Bmatrix} 400 \\ -950 \\ 400 \end{Bmatrix} = \begin{Bmatrix} -400 \\ 750 \\ -200 \end{Bmatrix}$$

and assign $\{D\}^0 = -\{R\}^0 = \begin{Bmatrix} 400 \\ -750 \\ 200 \end{Bmatrix}$

```

% Program ConGrad
% A program for solving a set of linear
% simultaneous equations [A]{X} = {B} by
% using Conjugate Gradient method.
% Read input data file containing number
% of eqs with matrix [A] and vector {B}:
fid = fopen('inputcg.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscsanf(fid, '%f', [n+1 n]); a = a.';
b = squeeze(a(:, n+1));
a = squeeze(a(:, 1:n));
fclose(fid);
xzero = 0.; x(1:n) = xzero;
x = CG(n, a, b, x);
% Print out the computed solution of {X}:
fid = fopen('output.dat', 'w');
fprintf(fid, ...
'\n Equation No.      Solution x \n');
fprintf(
...
'\n Equation No.      Solution x \n');
for i = 1:n
    fprintf(fid, '%8d %22.6e\n', i, x(i));
    fprintf(    '%8d %22.6e\n', i, x(i));
end
fclose(fid);

function x = CG(n, a, b, x)
% Assign initial values in vector {X}:
xzero = 0.; x(1:n) = xzero;
% Assign tolerance for stopping criterion:
tol = 0.0001;
% Compute the initial residual and search
% direction:
for i = 1:n
    sum = 0.;
    for j = 1:n
        sum = sum + a(i,j)*x(j);
    end
    r(i) = sum - b(i);
    d(i) = -r(i);
end
% Enter the iteration loop:
for k = 1:n+1
    up = 0.;
    for i = 1:n
        up = up + d(i)*r(i);
    end
    down = 0.;
    for i = 1:n
        sum = 0.;
        for j = 1:n
            sum = sum + a(i,j)*d(j);
        end
        down = down + d(i)*sum;
    end
    alam = -up/down;
    for i = 1:n
        x(i) = x(i) + alam*d(i);
    end
    for i = 1:n
        sum = 0.;
        for j = 1:n
            sum = sum + a(i,j)*x(j);
        end
        r(i) = sum - b(i);
    end
    res = 0.;
    for i = 1:n
        res = res + r(i)*r(i);
    end
    res = sqrt(res);
    if res < tol
        break
    end
    up = 0.;
    down = 0.;
    for i = 1:n
        sum = 0.;
        for j = 1:n
            sum = sum + a(i,j)*d(j);
        end
        up = up + r(i)*sum;
        down = down + d(i)*sum;
    end
    alpha = up/down;
    for i = 1:n
        d(i) = -r(i) + alpha*d(i);
    end
end

```

Figure 3.13 Computer program for solving the system of equations by using the conjugate gradient method.

Start the iteration process with $k = 0$,

$$\begin{aligned}
\lambda_0 &= - \frac{\lfloor D \rfloor^0 \{R\}^0}{\lfloor D \rfloor^0 [A] \{D\}^0} = 0.1260 \\
\{X\}^1 &= \{X\}^0 + \lambda_0 \{D\}^0 = \begin{cases} 150.4132 \\ 5.4752 \\ 125.2066 \end{cases} \\
\{R\}^1 &= [A]\{X\}^1 - \{B\} = \begin{cases} 179.7521 \\ 119.8347 \\ 89.8760 \end{cases} \\
Error &= \sqrt{\lfloor R \rfloor^1 \{R\}^1} = 233.9848 \\
\alpha_0 &= \frac{\lfloor R \rfloor^1 [A] \{D\}^0}{\lfloor D \rfloor^0 [A] \{D\}^0} = 0.0718
\end{aligned}$$

$$\{D\}^1 = -\{R\}^1 + \alpha_0 \{D\}^0 = \begin{Bmatrix} -151.0314 \\ -173.6861 \\ -75.5157 \end{Bmatrix}$$

Start the iteration process with $k = 1$,

$$\begin{aligned} \lambda_1 &= -\frac{\lfloor D \rfloor^1 \{R\}^1}{\lfloor D \rfloor^1 [A] \{D\}^1} = -1.9836 \\ \{X\}^2 &= \{X\}^1 + \lambda_1 \{D\}^1 = \begin{Bmatrix} 450.0000 \\ 350.0000 \\ 275.0000 \end{Bmatrix} \end{aligned}$$

The obtained solutions are equal to the exact solutions. The residual vector is,

$$\{R\}^2 = [A]\{X\}^2 - \{B\} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$$

So that $Error = \sqrt{\lfloor R \rfloor^2 \{R\}^2} = 0$ and the computation stops.

If the computation continues, then

$$\begin{aligned} \alpha_1 &= \frac{\lfloor R \rfloor^2 [A] \{D\}^1}{\lfloor D \rfloor^1 [A] \{D\}^1} = 0 \\ \text{and } \{D\}^2 &= -\{R\}^2 + \alpha_1 \{D\}^1 = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}. \end{aligned}$$

In addition, if the initial guess values in the vector $\lfloor X \rfloor^0 = [0 \ 0 \ 0]$ are used, the computational results from conjugate gradient method will converge to the solutions within 3 iterations ($k = 2$).

As the number of equations increases, the computational time from more numerical operations also increases. The number of operations required by the conjugate gradient method, however, can be reduced. For example, the residual vector $\{R\}$ at the $(k+1)^{th}$ iteration in Eq. (3.94) can be determined from the residual vector $\{R\}$ and the product of $[A]\{D\}$ from the k^{th} iteration. In addition, the residual vectors $\{R\}$ at different iterations have the orthogonal property, i.e.,

$$\lfloor R \rfloor^m \{R\}^n = 0 \quad \text{when } m \neq n \quad (3.101)$$

The property helps reducing the number of operations between matrices.

In conclusion, the modified conjugate gradient method that can reduce the computational time consists of determining the parameters as follows,

$$\lambda_k = \frac{\lfloor R \rfloor^k \{R\}^k}{\lfloor D \rfloor^k \{A\} \{D\}^k} \quad (3.102)$$

and

$$\alpha_k = \frac{\lfloor R \rfloor^{k+1} \{R\}^{k+1}}{\lfloor R \rfloor^k \{R\}^k} \quad (3.103)$$

Then, the following procedures are employed,

1. Assume the initial vector $\{X\}^0$ and calculate the residual vector from,

$$\{R\}^0 = [A]\{X\}^0 - \{B\}$$

2. Determine the search direction vector from,

$$\{D\}^0 = -\{R\}^0$$

3. Calculate the residual from,

$$\delta^0 = \lfloor R \rfloor^0 \{R\}^0$$

4. Start the iteration process from $k = 0, 1, 2, \dots, n$ by calculating,

$$\begin{aligned} \{U\}^k &= [A]\{D\}^k \\ \lambda_k &= \frac{\delta^k}{\lfloor D \rfloor^k \{U\}^k} \\ \{X\}^{k+1} &= \{X\}^k + \lambda_k \{D\}^k \\ \{R\}^{k+1} &= \{R\}^k + \lambda_k \{U\}^k \\ \delta^{k+1} &= \lfloor R \rfloor^{k+1} \{R\}^{k+1} \end{aligned}$$

Examine the error using $Error = \sqrt{\lfloor R \rfloor^{k+1} \{R\}^{k+1}} < \varepsilon$, where ε is the acceptable error. If it is true, the computation stops. If not, the iteration process continues by calculating,

$$\begin{aligned} \alpha_k &= \frac{\delta^{k+1}}{\delta^k} \\ \{D\}^{k+1} &= -\{R\}^{k+1} + \alpha_k \{D\}^k \end{aligned}$$

and then go back to the new k^{th} iteration in step 4.

The process explained above is used to develop a computer program as shown in Fig. 3.14.

Example 3.21 Use the modified conjugate gradient method in Fig. 3.14 to solve the system of equations,

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix} \quad (3.10)$$

Note that the matrix $[A]$ on left-hand side of Eq. (3.10) is an asymmetric matrix. The exact solutions for the above system of equations are $x_1 = 450$, $x_2 = 350$ and $x_3 = 275$.

```

% Program ConGradNew
% A program for solving a set of linear
% simultaneous equations [A]{X} = {B} by
% using Conjugate Gradient method.
% Read input data file containing number
% of eqs with matrix [A] and vector {B}:
fid = fopen('inputcg.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [n+1 n]); a = a.';
b = squeeze(a(:,1:n));
fclose(fid);
xzero = 0.; x(1:n) = xzero;
x = CGnew(n, a, b, x);
% Print out the computed solution of {X}:
fid = fopen('output.dat', 'w');
fprintf(fid, ...
'\n Equation No.      Solution x \n');
fprintf(
'\n Equation No.      Solution x \n');
for i = 1:n
    fprintf(fid, '%8d %22.6e\n', i, x(i));
    fprintf(' %8d %22.6e\n', i, x(i));
end
fclose(fid);

function x = CGnew(n, a, b, x)
% Assign initial values in vector {X}:
xzero = 0.; x(1:n) = xzero;
% Assign tolerance for stopping criterion:
tol = 0.0001;
% Compute the initial residual and search
% direction:
for i = 1:n
    sum = 0.;
    for j = 1:n
        sum = sum + a(i,j)*x(j);
    end
    r(i) = sum - b(i);
    d(i) = -r(i);
end
del = 0.;
for i = 1:n
    del = del + r(i)*r(i);
end
% Enter the interation loop:
for k = 1:n+1
    for i = 1:n
        u(i) = 0.;
        for j = 1:n
            u(i) = u(i) + a(i,j)*d(j);
        end
    end
    down = 0.;
    for i = 1:n
        down = down + d(i)*u(i);
    end
    alam = del/down;
    for i = 1:n
        x(i) = x(i) + alam*d(i);
        r(i) = r(i) + alam*u(i);
    end
    del2 = 0.;
    for i = 1:n
        del2 = del2 + r(i)*r(i);
    end
    if del2 < tol
        break
    end
    alpha = del2/del;
    for i = 1:n
        d(i) = -r(i) + alpha*d(i);
    end
    del = del2;
end

```

Figure 3.14 Computer program for solving the system of equations by the modified conjugate gradient method.

Solution The system of equations in Eq. (3.10) is in the form,

$$[P]\{X\} = \{Q\} \quad (3.104)$$

where $[P]$ is an asymmetric matrix. Equation (3.104) can be modified by pre-multiplying matrix $[P]^T$ on both sides so that the resulting matrix on the left-hand side of the equation becomes a symmetric matrix,

$$[P]^T[P]\{X\} = [P]^T\{Q\} \quad (3.105)$$

After pre-multiplying by matrix $[P]^T$, the new system of equations is

$$\begin{bmatrix} 17 & -20 & 2 \\ -20 & 36 & -16 \\ 2 & -16 & 20 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 1,200 \\ -800 \\ 800 \end{Bmatrix} \quad (3.106)$$

which is in the form of $[A]\{X\} = \{B\}$

The modified conjugate gradient method can then be applied by first assuming the initial guess vector, for example,

$$\{X\}^0 = \begin{Bmatrix} 100 \\ 100 \\ 100 \end{Bmatrix}$$

So that

$$\{R\}^0 = [A]\{X\}^0 - \{B\} = \begin{bmatrix} 17 & -20 & 2 \\ -20 & 36 & -16 \\ 2 & -16 & 20 \end{bmatrix} \begin{Bmatrix} 100 \\ 100 \\ 100 \end{Bmatrix} - \begin{Bmatrix} 1,200 \\ -800 \\ 800 \end{Bmatrix} = \begin{Bmatrix} -1,300 \\ 800 \\ -200 \end{Bmatrix}$$

Then

$$\{D\}^0 = -\{R\}^0 = \begin{Bmatrix} 1,300 \\ -800 \\ 200 \end{Bmatrix}$$

and

$$\delta^0 = \lfloor R \rfloor^0 \{R\}^0 = 2,370,000$$

By starting the iteration process at $k = 0$,

$$\{U\}^0 = [A]\{D\}^0 = \begin{Bmatrix} 38,500 \\ -58,000 \\ 19,400 \end{Bmatrix}$$

$$\lambda_0 = \frac{\delta^0}{\lfloor D \rfloor^0 \{U\}^0} = 0.0236$$

$$\{X\}^1 = \{X\}^0 + \lambda_0 \{D\}^0 = \begin{Bmatrix} 130.7087 \\ 81.1024 \\ 104.7244 \end{Bmatrix}$$

$$\{R\}^1 = \{R\}^0 + \lambda_0 \{U\}^0 = \begin{Bmatrix} -390.5512 \\ -570.0787 \\ 258.2677 \end{Bmatrix}$$

$$\delta^1 = \lfloor R \rfloor^1 \{R\}^1 = 544,222.1689$$

$$Error = \sqrt{\lfloor R \rfloor^1 \{R\}^1} = 737.7142$$

$$\alpha_0 = \frac{\delta^1}{\delta^0} = 0.2296$$

$$\{D\}^1 = -\{R\}^1 + \alpha_0 \{D\}^0 = \begin{Bmatrix} 689.0697 \\ 386.3750 \\ -212.3418 \end{Bmatrix}$$

For the iteration process at $k = 1$,

$$\{U\}^1 = [A]\{D\}^1 = \begin{Bmatrix} 3,562.0 \\ 3,525.6 \\ -9,050.7 \end{Bmatrix}$$

$$\begin{aligned}
\lambda_1 &= \frac{\delta^1}{\lfloor D \rfloor^1 \{U\}^1} & = & 0.0948 \\
\{X\}^2 &= \{X\}^1 + \lambda_1 \{D\}^1 & = & \begin{Bmatrix} 196.0579 \\ 117.7450 \\ 84.5866 \end{Bmatrix} \\
\{R\}^2 &= \{R\}^1 + \lambda_1 \{U\}^1 & = & \begin{Bmatrix} -52.7418 \\ -235.7237 \\ -600.0730 \end{Bmatrix} \\
\delta^2 &= \lfloor R \rfloor^2 \{R\}^2 & = & 418,434.9655 \\
Error &= \sqrt{\lfloor R \rfloor^2 \{R\}^2} & = & 646.8655 \\
\alpha_1 &= \frac{\delta^2}{\delta^1} & = & 0.7689 \\
\{D\}^2 &= -\{R\}^2 + \alpha_1 \{D\}^1 & = & \begin{Bmatrix} 582.5454 \\ 532.7951 \\ 436.8102 \end{Bmatrix}
\end{aligned}$$

For the iteration process at $k = 2$,

$$\begin{aligned}
\{U\}^2 &= [A] \{D\}^2 & = & \begin{Bmatrix} 120.9903 \\ 540.7525 \\ 1,376.5731 \end{Bmatrix} \\
\lambda_2 &= \frac{\delta^2}{\lfloor D \rfloor^2 \{U\}^2} & = & 0.435918 \\
\{X\}^3 &= \{X\}^2 + \lambda_2 \{D\}^2 & = & \begin{Bmatrix} 450.0000 \\ 350.0000 \\ 275.0000 \end{Bmatrix} \\
\{R\}^3 &= \{R\}^2 + \lambda_2 \{U\}^2 & = & \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \\
\delta^3 &= \lfloor R \rfloor^3 \{R\}^3 & = & 0 \\
Error &= \sqrt{\lfloor R \rfloor^3 \{R\}^3} & = & 0
\end{aligned}$$

This example demonstrates that the computed solutions from the modified conjugate gradient method converge to the exact solutions of $x_1 = 450$, $x_2 = 350$ and $x_3 = 275$ with the error of $\delta = 0$.

For practical problems that are governed by nonlinear differential equations, the derived matrix $[A]$ is usually an asymmetric and non-positive definite matrix. In this case, the conjugate gradient method presented herein must be modified. The modification leads to the new methods such as the generalized minimal residual method or GMRES, the bi-conjugate gradient method, the squared conjugate gradient method and the quasi-minimal residual method, etc. Details for these methods are beyond the scope of this book but can be found in research literatures.

3.17 Closure

In this chapter, various methods for solving a set of system of equations, $[A]\{X\} = \{B\}$, are presented. These methods are classified into two groups of the direct and iterative techniques. The direct technique consists of the Cramer's rule, the Gauss-elimination method, the Gauss-Jordan method, the matrix inversion method, the LU decomposition method and the Cholesky method. The Cramer's rule should not be used in general because it requires a large number of operations as compared to the others. The Guass-elimination and the LU decomposition methods are popular and widely used for solving practical problems. The Cholesky decomposition method is often used for solving many engineering problems where the matrix $[A]$ is symmetric.

The iterative technique consists of the Jacobi method, the Gauss-Seidel method, the successive over-relaxation method and the conjugate gradient method. The conjugate gradient method is considered as a popular method today for solving large size problems because of its high computational efficiency.

The computational procedures and examples presented in this chapter can help readers to understand the concepts of various methods easily. The accompanied computer programs of each method can also be used for solving a large set of system of equations. Some of these computer programs can be implemented directly into the finite difference or finite element software for analyzing practical engineering problems.

Exercises

1. Solve the following system of equations by using the Cramer's rule,

$$\begin{aligned}-2x_1 + 3x_2 + x_3 &= 9 \\ 3x_1 + 4x_2 - 5x_3 &= 0 \\ x_1 - 2x_2 + x_3 &= -4\end{aligned}$$

2. Solve the following system of equations by using the Cramer's rule,

$$\begin{bmatrix} 1 & 1 & 3 \\ 5 & 3 & 1 \\ 2 & 3 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 2 \\ 3 \\ -1 \end{Bmatrix}$$

3. Solve the following system of equations by using the Cramer's rule,

$$\begin{bmatrix} 2 & 3 & 5 \\ 3 & 1 & -2 \\ 1 & 3 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -2 \\ -3 \end{Bmatrix}$$

4. Solve the system of equations in Problems 1-3 by using the left division technique in MATLAB. Verify the computed solutions by substituting them back into the system of equations.

5. Show detailed computational procedure for solving the following system of equations,

$$\begin{bmatrix} 1 & 2 & 4 \\ 4 & -1 & 1 \\ 2 & 5 & 2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 11 \\ 8 \\ 3 \end{Bmatrix}$$

by using

- (a) the Gauss elimination method with scaling and pivoting.
- (b) the LU decomposition method.

Verify the computed solutions by substituting them back into the system of equations.

6. Solve the following system of equations by using the Gauss elimination method,

$$\begin{aligned} 3x_1 - 2x_2 + 8x_3 &= 21 \\ -5x_1 + 6x_2 + 4x_3 &= 19 \\ 8x_1 + 3x_2 - 5x_3 &= -1 \end{aligned}$$

- (a) without pivoting and scaling.
- (b) with pivoting and scaling.
- (c) verify the computed solutions by using the computer program in Fig. 3.3.

7. Solve the following system of equations by using the Gauss elimination method,

$$\begin{bmatrix} 1 & 2 & 1 & 4 \\ 4 & 2 & 0 & 3 \\ 3 & 0 & 2 & 1 \\ 0 & 3 & 1 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 12.700 \\ 11.880 \\ 18.218 \\ 17.100 \end{Bmatrix}$$

Verify the computed solutions with Eq. (2.59).

8. Show detailed computational procedure for solving the following system of equations,

$$\begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{Bmatrix}$$

by using: (a) the Gauss elimination method, (b) the LU decomposition method, and (c) the Cholesky decomposition method. Verify the solutions by using one of the computer programs presented in this chapter.

9. Solve the system of equations shown in Problem 5 by using the matrix inversion that is based on the Gauss Jordan method. Show detailed computational procedure. Explain advantages and disadvantages of the method.

10. Solve the following system of equations,

$$\begin{bmatrix} 0 & -1 & 3 \\ 4 & 0 & -1 \\ -2 & 5 & 0 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 3 \\ 14 \\ 7 \end{Bmatrix}$$

by using: (a) the Gauss elimination method, (b) the LU decomposition method, (c) the Gauss-Seidel iteration method, and (d) the conjugate gradient method. Explain difficulties that arise in each method and suggest how to overcome them.

11. Use the Gauss elimination method to solve the following system of equations,

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 2 & 3 & 4 \\ 3 & 3 & 3 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 1.234 \\ 2.234 \\ 3.334 \\ 4.444 \end{Bmatrix}$$

Compare the computed solutions with those obtained from the Gauss-Seidel method that uses the stopping tolerance of $\varepsilon = 0.000001\%$. Then explain advantages and disadvantages of both methods.

12. The exact solutions of the following system of equations,

$$\begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 11/6 \\ 13/12 \\ 47/60 \end{Bmatrix}$$

are $x_1 = x_2 = x_3 = 1$. Use the Gauss elimination method to solve for the solutions. Explain why the computed solutions contain errors. Then suggest methods to reduce such errors and use them to improve the solutions.

13. Modify the Gauss elimination method to solve the tri-diagonal system of equations in the form,

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & \ddots & \ddots & \ddots & 0 & 0 \\ 0 & 0 & \ddots & \ddots & \ddots & 0 \\ 0 & 0 & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & a_n & b_n \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{Bmatrix} = \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{Bmatrix}$$

Show detailed computational procedure with a corresponding computer program.

14. Solve the following tri-diagonal system of equations

$$\begin{bmatrix} 4 & -3 & & & & \\ -3 & 2 & 1 & & & \\ & 1 & 3 & -1 & & \\ & & -1 & 5 & -4 & \\ & & & -4 & 2 & \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 4 \\ 7 \\ -3 \\ -6 \end{Bmatrix}$$

Verify the computed solutions by using the computer program in Fig. 3.4.

15. Use the Gauss Jordan method to find the inverses of the following matrices,

$$(a) \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & -1 \\ 2 & 1 & 7 \end{bmatrix} ; \quad (b) \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 3 & 3 & 7 \end{bmatrix}$$

Then verify the results with those obtained by using the Cramer's rule.

16. Find the inverse of the matrix,

$$[A] = \begin{bmatrix} 1 & 1/2 & 1/3 \\ 1/2 & 1/3 & 1/4 \\ 1/3 & 1/4 & 1/5 \end{bmatrix}$$

by using the Gauss Jordan method as explained in section 3.7. Examine whether the matrix is ill-conditioned by using the three criteria as explained at the end of section 3.7.

17. Use the Cholesky decomposition computer program as shown in Fig. 3.6 to solve the following system of equations,

$$\begin{bmatrix} 4 & -4 & 0 \\ -4 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ -950 \\ 400 \end{Bmatrix}$$

which has exact solutions of $x_1 = 450$, $x_2 = 350$ and $x_3 = 275$. Explain difficulties that may arise by using the method and how to overcome them.

18. Study the expressions for obtaining the coefficients in the matrices $[L]$ and $[U]$ as shown in Eqs. (3.55a-e) of the LU decomposition method. Prepare these expressions for the system of 4 equations. Then, follow Eqs. (3.56a-b) to write expressions for the forward and back substitution procedures. Use the system of 4 equations in Problem 7 to verify the expressions derived.
19. Add the pivoting and scaling capability as explained in sections 3.5.1-2 to the computer program that uses the LU decomposition method for solving the system of equations shown in Fig. 3.5. Then use the modified program to solve the system of equations in Problem 17.
20. Develop a compute program for solving the system of nonlinear equations by the Newton-Raphson method. During the iteration procedure, use the Gauss elimination method as shown by the computer program in Fig. 3.3 to solve the system of equations. Verify the developed computer program by solving Example 2.12 and compare the solutions with those shown in Table 2.8.
21. Develop a computer program to solve the following system of equations,

$$\begin{aligned} 4x + y^2 + z &= 11 \\ x + 4y + z^2 &= 18 \\ x^2 + y + 4z &= 15 \end{aligned}$$

by using the Gauss-Seidel iteration and conjugate gradient methods. Use the initial guess values of $x = y = z = 1$ with the acceptable error of $\varepsilon = 0.000001\%$.

22. Develop a computer program to solve the following system of equations,

$$\begin{bmatrix} 3 & 2 & 0 & 0 \\ 2 & 3 & 2 & 0 \\ 0 & 2 & 3 & 2 \\ 0 & 0 & 2 & 3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 12 \\ 17 \\ 14 \\ 7 \end{Bmatrix}$$

by using: (a) the Jacobi iteration method, (b) the Gauss-Seidel iteration method, (c) the successive over-relaxation method with $1.20 \leq \omega \leq 1.40$ for every $\Delta\omega = 0.01$, and (d) the conjugate gradient method with the acceptable error of $\varepsilon = 0.001\%$.

23. Study the convergence rates for the following system of equations,

$$\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{Bmatrix}$$

by using: (a) the Gauss-Seidel iteration method, (b) the successive over-relaxation method with $\omega = 1.25, 1.50, 1.75$, and (c) the conjugate gradient method with the initial guess values of $x_1 = x_2 = x_3 = x_4 = 0$. Compare the computed solutions with those obtained from the Gauss elimination method.

24. Study the convergence rate for solving the system of Eq. (3.10) by employing the successive over-relaxation method. Use the weighting factor ω that varies from 0 to 2 with the increment of 0.05. Then, suggest the optimal value of ω for solving this system of equations.
25. Solve the system of equations in Problem 12 again but by using the conjugate gradient method. Use appropriate initial guess values to obtain the converged solutions.
26. Solve the following system of equations by using the conjugate gradient method,

$$\begin{bmatrix} 2 & 4 & 1 \\ 4 & 3 & 2 \\ 1 & 2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 13 \\ 16 \\ 17 \end{Bmatrix}$$

with the initial guess values of $x_1 = x_2 = x_3 = 1$. Show the computational procedures in details.

27. Matrix $[A]$ in the following system of equations is an asymmetric matrix,

$$\begin{bmatrix} 1 & 2 & 3 \\ -2 & 1 & 4 \\ 1 & -3 & 2 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 8 \\ 4 \\ -1 \end{Bmatrix}$$

Use the conjugate gradient method to solve for the solutions with the initial guess values of $x_1 = x_2 = x_3 = 1$. Show the computational procedures in details.

28. Solve the following system of equations,

$$\begin{bmatrix} 17 & 1 & 4 & 3 & -1 & 2 & 3 & -7 \\ 2 & 10 & -1 & 1 & -2 & 1 & 1 & -4 \\ -1 & 1 & -8 & 2 & -5 & 2 & -1 & 1 \\ 2 & 4 & 1 & -11 & 1 & 3 & 4 & -1 \\ 1 & 3 & 1 & 7 & -15 & 1 & -2 & 4 \\ -2 & 1 & 7 & -1 & 2 & 12 & -1 & 8 \\ 3 & 4 & 5 & 1 & 2 & 8 & -19 & 2 \\ 5 & 1 & 1 & 1 & 1 & 1 & -7 & 10 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{Bmatrix} = \begin{Bmatrix} 71 \\ 43 \\ -11 \\ -37 \\ -61 \\ 52 \\ -73 \\ 21 \end{Bmatrix}$$

by using the computer programs of:

- (a) the Gauss elimination method
- (b) the LU decomposition method
- (c) the conjugate gradient method

presented in this chapter. Provide comments on the accuracy of the solutions obtained from the three methods.

29. Solve the following system of equations,

$$\begin{bmatrix} 8 & -2 & 1 & 0 & \dots & \dots & \dots & \dots & \dots & 0 \\ -2 & 8 & -2 & 1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 1 & -2 & 8 & -2 & 1 & 0 & \dots & \dots & \dots & 0 \\ 0 & 1 & -2 & 8 & -2 & 1 & 0 & \dots & \dots & 0 \\ \dots & \dots \\ \dots & \dots \\ 0 & \dots & \dots & 0 & 1 & -2 & 8 & -2 & 1 & 0 \\ 0 & \dots & \dots & \dots & 0 & 1 & -2 & 8 & -2 & 1 \\ 0 & \dots & \dots & \dots & \dots & 0 & 1 & -2 & 8 & -2 \\ 0 & \dots & \dots & \dots & \dots & \dots & 0 & 1 & -2 & 8 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \dots \\ \dots \\ x_{47} \\ x_{48} \\ x_{49} \\ x_{50} \end{Bmatrix} = \begin{Bmatrix} 100 \\ 100 \\ 100 \\ 100 \\ \dots \\ \dots \\ 100 \\ 100 \\ 100 \\ 100 \end{Bmatrix}$$

by using the computer programs of:

- (a) the Gauss elimination method
- (b) the LU decomposition method
- (c) the Cholesky decomposition method
- (d) the conjugate gradient method

presented in this chapter. Explain advantages and disadvantages of each method for solving such system of equations.

30. Solve the system of equations in Problem 28 again but by using the function `lu` with the left division technique in MATLAB. Compare the computed solutions with those obtained in Problem 28.
31. Solve the system of equations in Problem 29 again but using the function `lu` or `chol` with the left division technique in MATLAB. Compare the computed solutions with those obtained in Problem 29.
32. In the determination of the currents and voltages in an electrical circuit in Fig. P3.32, the Kirchhoff's current rule (summation of currents at any point is zero) and the Kirchhoff's voltage rule (summation of voltages in any closed-circuit loop is zero) are needed.

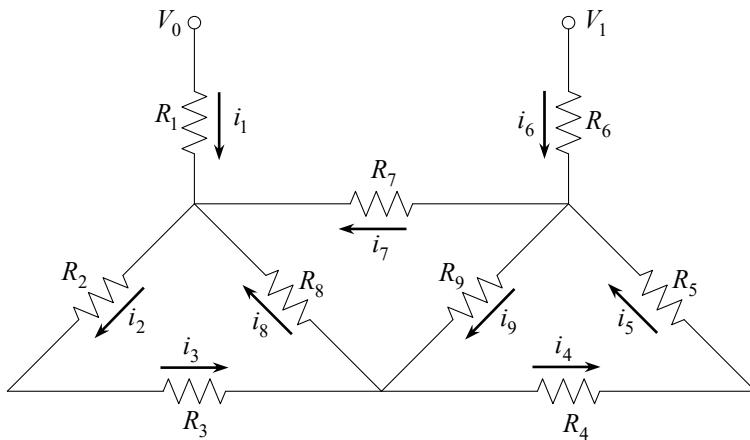


Figure P3.32.

By applying the Kirchhoff's current and voltage rules on the circuit, the following system of equations is obtained,

$$\begin{array}{ll}
 i_2 - i_3 = 0 & i_2 R_2 + i_3 R_3 + i_8 R_8 = 0 \\
 i_4 - i_5 = 0 & i_7 R_7 - i_8 R_8 - i_9 R_9 = 0 \\
 i_1 + i_7 + i_8 - i_2 = 0 & i_9 R_9 + i_4 R_4 + i_5 R_5 = 0 \\
 i_3 + i_9 - i_8 - i_4 = 0 & i_1 R_1 - i_7 R_7 - i_6 R_6 = V_0 - V_1 \\
 i_6 + i_5 - i_7 - i_9 = 0 &
 \end{array}$$

If \$V_0 = 150\$ Volt, \$V_1 = 0\$ Volt, \$R_1 = R_8 = R_9 = 10 \Omega\$, \$R_2 = R_4 = 5 \Omega\$, \$R_3 = R_6 = 15 \Omega\$ and \$R_5 = R_7 = 20 \Omega\$, use MATLAB to solve the system of equations. Compare the solutions with those obtained by using the Gauss elimination method.

33. Develop a system of equations by apply the Kirchhoff's current and voltage rules on the electrical circuit in Fig. P3.33. Then, use MATLAB to solve such system of equations to obtain the currents passing through each resistor. Compare the computed solutions with those obtained from using the Gauss-Seidel iteration method.

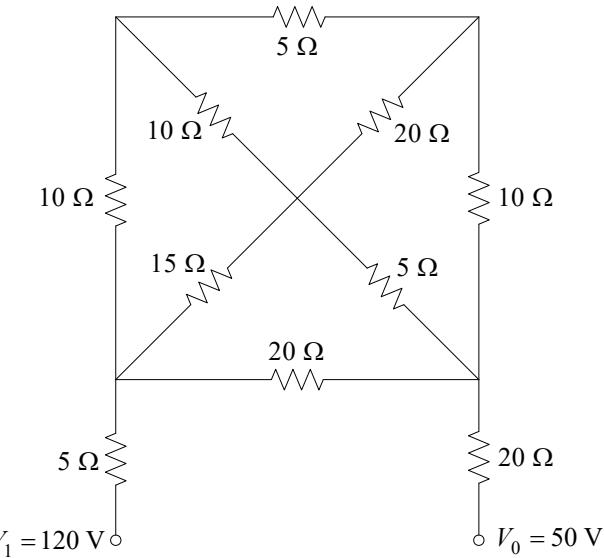


Figure P3.33.

34. A moving system of three weights connected by the two cords is shown in Fig. P3.34. The friction coefficients between the two weights on the floor are 0.3 and 0.5 as shown in the figure.

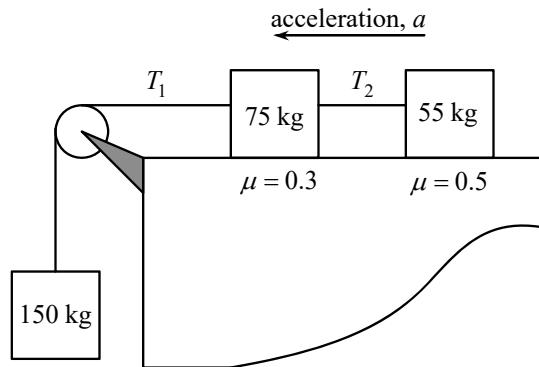


Figure P3.34.

By applying the Newton's second law ($\sum \bar{F} = m\bar{a}$) on the three weights, the following system of equations is obtained,

$$150a + T_1 = 1,471.500$$

$$75a - T_1 + T_2 = -220.725$$

$$55a - T_2 = -269.775$$

Use MATLAB to solve the system of equations for the system acceleration a and the tensions T_1 and T_2 . Compare the computed solutions with those obtained by using the Gauss elimination method.

35. A moving system consisting of three weights connected by cords is shown in Fig. P3.35. Apply the Newton's second law ($\sum \bar{F} = m\bar{a}$) on the three weights to derive a system of three equations. Then, use MATLAB to solve for the system acceleration a and the cord tensions T_1 and T_2 .

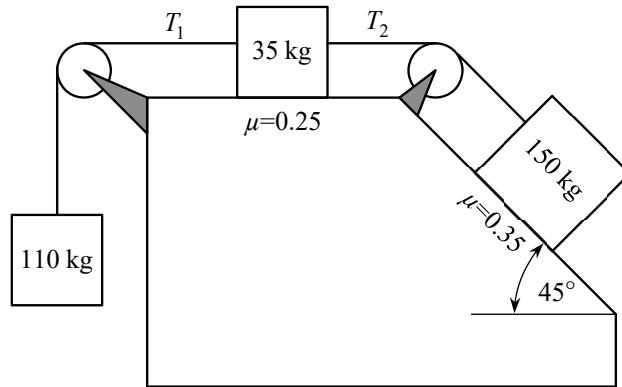


Figure P3.35.

Chapter

4

Interpolation and Extrapolation

4.1 Introduction

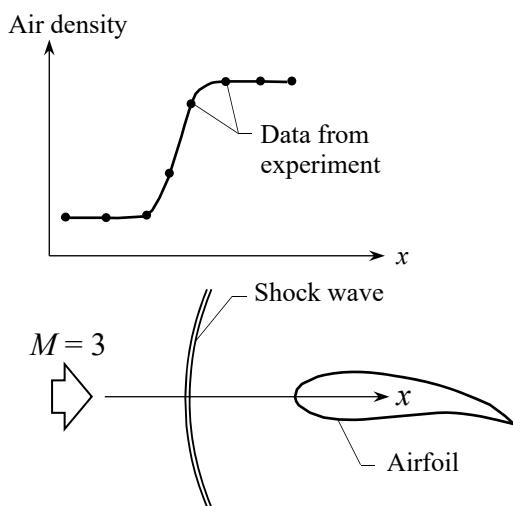


Figure 4.1 Shock wave and air density distribution in front of an airfoil.

Most of data obtained from experiment are always at discrete points. For examples, the temperatures on the solar heating panel are measured at specific positions or the pressures over an airfoil are collected at discrete locations. Often, the data at the other locations that have not been measured from the experiment are needed. Interpolation and extrapolation techniques are thus required to provide information at these locations.

Figure 4.1 shows experimental data of the air density from a supersonic Mach 3 flow over an airfoil. The data are collected at some discrete locations. It would be better if these data are fitted to yield a continuous function. The function can be used to provide information of the density at the other locations that have not been measured.

Another example that requires fitting a set of data is the prediction of the temperature through the thickness of a hot water pipe. The analytical solution of the temperature distribution

through the pipe thickness is in the form of the Bessel function. Values of the Bessel function is not easy to calculate so that they appear in form of a table in most of the heat transfer textbooks. Table 4.1 shows typical values of the Bessel function that varies with x . These values are at discrete points from $x = 2.0$ to 4.0 with an increment of 0.2. If the Bessel function at $x = 2.5784$ is needed, the data in the table must be interpolated. Many interpolation methods are explained in this chapter. These interpolation methods are basis for studying higher-level numerical methods, such as the finite difference and finite element methods.

Table 4.1 Values of the Bessel function at discrete points x .

x	2.0	2.2	2.4	2.6	2.8	3.0
$J_0(x)$	0.2239	0.1104	0.0025	-0.0968	-0.1850	-0.2601
x	3.0	3.2	3.4	3.6	3.8	4.0
$J_0(x)$	-0.2601	-0.3202	-0.3643	-0.3918	-0.3992	-0.3971

The widely used interpolation methods are: (1) the Newton's divided differences, (2) the Lagrange polynomials, and (3) the spline interpolation. These methods are presented and explained in details herein.

4.2 Newton's Divided Differences

One of the most popular methods for interpolating a set of data points is the Newton's divided differences. The main idea of the method is to create a polynomial function $f(x)$ from the given data points and use that function to interpolate the values at any other locations. The simplest form of such function is linear as explained in the following section.

4.2.1 Linear interpolation

From Table 4.1, if the value of the Bessel function at any point x between the two points $x_0 = 2.0$ and $x_1 = 4.0$ is needed, the easiest way to estimate such value is to create a linear function or the first-order polynomial between these two data points as shown by the dashed line in Fig. 4.2. The dashed line is then used to estimate the value of Bessel function at the point x from the values of the function at x_0 and x_1 .

The first-order polynomial or linear function can be derived by writing it in the form

$$f(x) = C_0 + C_1(x - x_0) \quad (4.1)$$

where C_0 and C_1 are unknown constants that can be determined by using the conditions at the points x_0 and x_1 as follows,

$$\text{at } x = x_0 : \quad f(x_0) = C_0 + 0 = C_0 \quad (4.2a)$$

$$\begin{aligned} \text{at } x = x_1 : \quad f(x_1) &= C_0 + C_1(x_1 - x_0) \\ &= f(x_0) + C_1(x_1 - x_0) \\ C_1 &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned} \quad (4.2b)$$

Thus,

$$f(x) = f(x_0) + (x - x_0) \frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad (4.3)$$

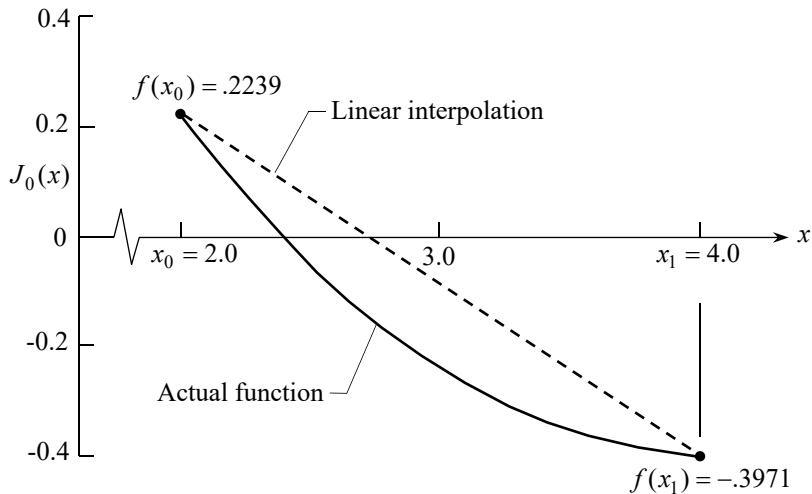


Figure 4.2 Use of a linear function to interpolate the Bessel function values between $x_0 = 2.0$ and $x_1 = 4.0$.

Example 4.1 The values of the Bessel function $f(x_0 = 2.0) = 0.2239$ and $f(x_1 = 4.0) = -0.3971$ are given at two points in Table 4.1 as shown in Fig. 4.2. Estimate the value of the Bessel function at the point $x = 3.2$ by using the linear interpolation. Compare the estimated value with the exact solution shown in Table 4.1.

From Eq. (4.3), the estimated value of the Bessel function at the point $x = 3.2$ by using the linear interpolation is

$$\begin{aligned} f(x = 3.2) &= 0.2239 + (3.2 - 2.0) \frac{-0.3971 - 0.2239}{4.0 - 2.0} \\ &= 0.2239 - 0.3726 \\ f(3.2) &= -0.1487 \end{aligned} \tag{4.4}$$

The estimated value has the true error of

$$\varepsilon = \frac{-0.3202 + 0.1487}{-0.3202} \times 100\% = 53.56\% \tag{4.5}$$

4.2.2 Quadratic interpolation

A quadratic interpolation can be derived by using the same procedure as for the linear interpolation explained in the preceding section. The quadratic interpolation is based on the second-order polynomial. Their coefficients are determined by using the three data values at x_0 , x_1 and x_2 . The quadratic function that passes through the three data values is written in the form

$$f(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) \tag{4.6}$$

where C_0 , C_1 and C_2 are unknown constants which can be determined by using the three data values at x_0 , x_1 and x_2 . If the three data values are $f(x_0 = 2.0) = 0.2239$, $f(x_1 = 3.0) = -0.2601$ and $f(x_2 = 4.0) = -0.3971$, the unknown constants can be determined as follows

$$\text{at } x = x_0 : \quad f(x_0) = C_0 + 0 + 0 = C_0 \quad (4.7a)$$

$$\begin{aligned} \text{at } x = x_1 : \quad f(x_1) &= C_0 + C_1(x_1 - x_0) \\ &= f(x_0) + C_1(x_1 - x_0) \end{aligned}$$

$$C_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad (4.7b)$$

$$\text{at } x = x_2 : \quad f(x_2) = C_0 + C_1(x_2 - x_0) + C_2(x_2 - x_0)(x_2 - x_1)$$

$$C_2 = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} \quad (4.7c)$$

These values of C_0 , C_1 and C_2 are then substituted into Eq. (4.6) to obtain the quadratic interpolation.

Example 4.2 Three data values of the Bessel function $f(x_0 = 2.0) = 0.2239$, $f(x_1 = 3.0) = -0.2601$ and $f(x_2 = 4.0) = -0.3971$ are given in Table 4.1 as shown in Fig. 4.3. Estimate the value of the Bessel function at $x = 3.2$ by using the quadratic interpolation. Compare the estimated value with exact solution in Table 4.1.

From the given three data points, the constants C_1 , C_2 and C_3 are determined from Eq. (4.7a-c) as

$$C_0 = 0.2239$$

$$C_1 = \frac{-0.2601 - 0.2239}{3.0 - 2.0} = -0.4840$$

$$C_2 = \frac{\frac{-0.3971 + 0.2601}{4.0 - 3.0} - \frac{-0.2601 - 0.2239}{3.0 - 2.0}}{4.0 - 2.0} = 0.1735$$

By substituting these constants into Eq. (4.6), the quadratic interpolation is

$$f(x) = 0.2239 - 0.4840(x - 2.0) + 0.1735(x - 2.0)(x - 3.0)$$

Then, the Bessel function at $x = 3.2$ can be determined as

$$f(3.2) = 0.2239 - 0.4840(1.2) + 0.1735(1.2)(0.2)$$

$$= 0.2239 - 0.5808 + 0.0416$$

$$f(3.2) = -0.3153 \quad (4.8)$$

The true error as compared to the exact solution is

$$\varepsilon = \frac{-0.3202 + 0.3153}{-0.3202} \times 100\% = 1.53\% \quad (4.9)$$

which is less than the error obtained from the linear interpolation in Example 4.1.

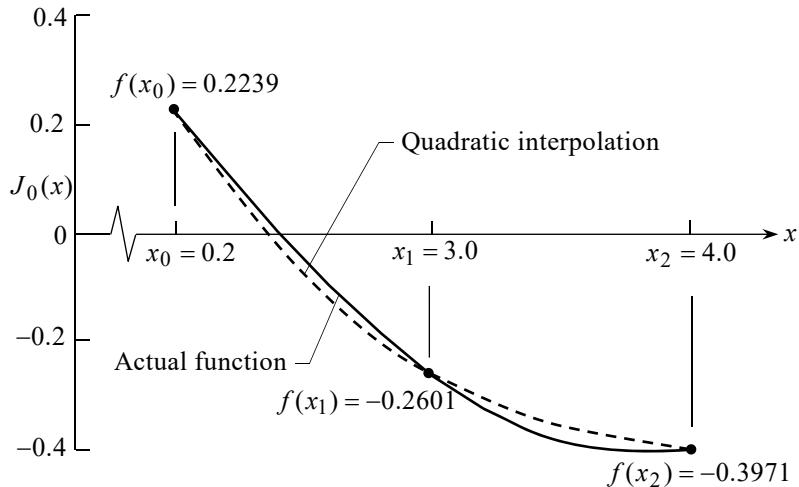


Figure 4.3 Use of a quadratic function to interpolate the Bessel function values between $x_0 = 2.0$ and $x_2 = 4.0$.

4.2.3 n^{th} -order Polynomial interpolation

The procedure explained in the preceding sections can be used to create the n^{th} -order polynomial interpolation passing through the $n+1$ data values at $x_0, x_1, x_2, \dots, x_n$ as shown in Fig. 4.4.

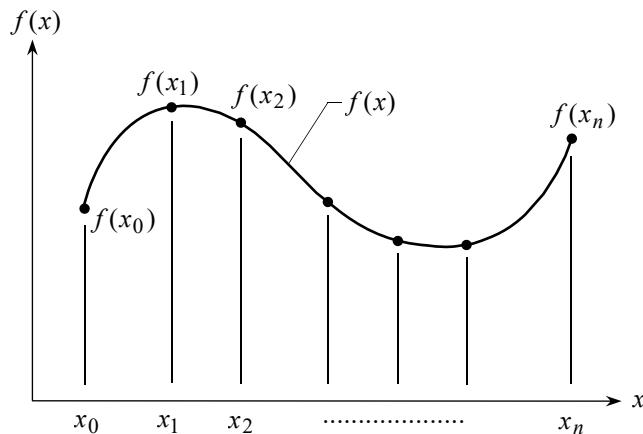


Figure 4.4 The n^{th} -order polynomial interpolation passing through $n+1$ data values.

The n^{th} -order polynomial interpolation as shown in Fig. 4.4 can be written in a general form as

$$\begin{aligned} f(x) &= C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + C_3(x - x_0)(x - x_1)(x - x_2) \\ &\quad + \dots + C_n(x - x_0)(x - x_1)\dots(x - x_{n-1}) \end{aligned} \quad (4.10)$$

where the coefficients C_i , $i = 0, 1, 2, \dots, n$ can be determined and are obtained in the forms

$$C_0 = f(x_0) \quad (4.11a)$$

$$C_1 = f[x_1, x_0] \quad (4.11b)$$

$$C_2 = f[x_2, x_1, x_0] \quad (4.11c)$$

$$\vdots \quad \vdots \quad \vdots$$

$$C_n = f[x_n, x_{n-1}, \dots, x_1, x_0] \quad (4.11n)$$

The square parentheses shown in Eq. (4.11) represent the divided differences. For example, the first divided difference is determined from

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j} \quad (4.12)$$

The result is the same as the value of C_1 in Eq. (4.7b). The second divided difference is

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k} \quad (4.13)$$

which is in the same form of Eq. (4.7c) for determining the value of C_2 . Thus, the n^{th} divided difference can be written in the form

$$f[x_n, x_{n-1}, \dots, x_1, x_0] = \frac{f[x_n, x_{n-1}, \dots, x_1] - f[x_{n-1}, \dots, x_1, x_0]}{x_n - x_0} \quad (4.14)$$

The values of divided differences as shown in Eqs. (4.12) - (4.14) can be determined sequentially as shown in Table 4.2.

Table 4.2 Sequential steps to determine the values of the divided differences.

<i>i</i>	<i>x</i>	<i>f(x)</i>	Divided differences		
			First	Second	Third
0	x_0	$f(x_0)$	$f[x_1, x_0]$	$f[x_2, x_1, x_0]$	$f[x_3, x_2, x_1, x_0]$
1	x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	
2	x_2	$f(x_2)$	$f[x_3, x_2]$		
3	x_3	$f(x_3)$			

The divided differences in the first row of Table 4.2 represent the values of the constants C_i , $i = 0, 1, 2, \dots, n$ as shown in Eq. (4.11). By substituting these divided differences into Eq. (4.10), the general form of the n^{th} -order polynomial interpolation is

$$\begin{aligned} f(x) &= f(x_0) + (x - x_0)f[x_1, x_0] + (x - x_0)(x - x_1)f[x_2, x_1, x_0] \\ &\quad + \dots + (x - x_0)(x - x_1)\dots(x - x_{n-1})f[x_n, x_{n-1}, \dots, x_0] \end{aligned} \quad (4.15)$$

Equation (4.15) is called the Newton's divided-difference interpolating polynomials. The equation can be used to develop a corresponding computer program directly.

Figure 4.5 shows a computer program to determine a value from a set of data points by using the Newton's divided-difference interpolating polynomials in Eq. (4.15).

```
% Program NewDiv
% Program for computing f(x) at a given x
% by using Newton's divided-difference
% interpolating polynomials.
% Read number of data set:
fid = fopen('input.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [2 n]);
x = a(1,:); x = x';
fx = a(2,:); fx = fx';
fclose(fid);
% Compute divided-difference coefficients:
m = n;
for icol = 2:n
    m = m-1;
    for irow = 1:m
        % Compute divided-difference coefficients:
        m = n;
        for icol = 2:n
            m = m-1;
            for irow = 1:m
                % Compute divided-difference coefficients:
                m = n;
                for icol = 2:n
                    m = m-1;
                    for irow = 1:m
                        % Compute divided-difference coefficients:
                        m = n;
                        for icol = 2:n
                            m = m-1;
                            for irow = 1:m
                                % Compute divided-difference coefficients:
                                m = n;
                                for icol = 2:n
                                    m = m-1;
                                    for irow = 1:m
                                        % Compute divided-difference coefficients:
                                        m = n;
                                        for icol = 2:n
                                            m = m-1;
                                            for irow = 1:m
                                                % Compute divided-difference coefficients:
                                                m = n;
                                                for icol = 2:n
                                                    m = m-1;
                                                    for irow = 1:m
                                                        % Compute divided-difference coefficients:
                                                        m = n;
                                                        for icol = 2:n
                                                            m = m-1;
                                                            for irow = 1:m
                                                                % Compute divided-difference coefficients:
                                                                m = n;
                                                                for icol = 2:n
                                                                    m = m-1;
                                                                    for irow = 1:m
                                                                        % Compute divided-difference coefficients:
                                                                        m = n;
                                                                        for icol = 2:n
                                                                            m = m-1;
                                                                            for irow = 1:m
                                                                                % Compute divided-difference coefficients:
                                                                                m = n;
                                                                                for icol = 2:n
                                                                                    m = m-1;
                                                                                    for irow = 1:m
                                                                                        % Compute divided-difference coefficients:
                                                                                        m = n;
                                                                                        for icol = 2:n
                                                                                            m = m-1;
                                                                                            for irow = 1:m
                                                                                                % Compute divided-difference coefficients:
                                                                                                m = n;
                                                                                                for icol = 2:n
                                                                                                 m = m-1;
                                                                                                 for irow = 1:m
                                                                                                     % Compute divided-difference coefficients:
                                                                                                     m = n;
                                                                                                     for icol = 2:n
                                                                                                         m = m-1;
................................................................
```

Figure 4.5 A computer program for determining an interpolated value from a given set of data points by using the Newton's divided-difference method.

Example 4.3 Employ the computer program in Fig. 4.5 of the Newton's divided-difference method to estimate the value of the Bessel function at $x = 3.2$. Use all data points in Table 4.1 except the data at $x = 3.2$. Then, determine the true error by comparing the estimated value with the exact solution.

Figure 4.6 shows the input data needed by the computer program. The input data consists of all data values that appear in Table 4.1 except the value at $x = 3.2$. The estimated value obtained from the computer program is -0.3201 . Thus, the true error is

$$\varepsilon = \frac{-0.3202 + 0.3201}{-0.3202} \times 100\% = 0.03\% \quad (4.16)$$

10	
2.0	.2239
2.2	.1104
2.4	.0025
2.6	-.0968
2.8	-.1850
3.0	-.2601
3.4	-.3643
3.6	-.3918
3.8	-.3992
4.0	-.3971
3.2	
Value of f(x) at x = .3200E+01 is -.3201119E+00	

Figure 4.6 Input data from Table 4.1 for the computer program in Fig. 4.5 that uses the Newton's divided-difference method and the computed value.

The estimated value at $x = 3.2$ obtained from using the values at the ten data points is quite accurate with the error of only 0.03%. It is noted that the errors of the estimated values by using the linear and quadratic interpolations in Examples 4.1 and 4.2 are 53.56% and 1.53%, respectively.

The above examples show that Newton's divided-difference method is simple and easy to understand. The computational procedure of the method can be used to develop a computer program directly. The main idea of the method is to derive a function that passes through all data points. The function is then used to determine values at locations within these data points. The general form of the Newton's divided-difference interpolating polynomials consists of the set of coefficient C_i as shown in Eq. (4.10). These coefficients can be determined by using the sequential steps as explained in Table 4.2. In the following section, another method that can be used to interpolate values from the given data points is presented. The method is also simple and represents a basis to study high-level numerical methods for solving practical problems.

4.3 Lagrange Interpolating Polynomials

The Lagrange polynomials are widely used to interpolate values from a set of data points. The first-order Lagrange polynomial is the simplest one which uses a linear function as explained below.

4.3.1 Linear interpolation

A straight line is used to connect the two data values of $f(x_0)$ and $f(x_1)$ at x_0 and x_1 , respectively, as shown in Fig. 4.7.

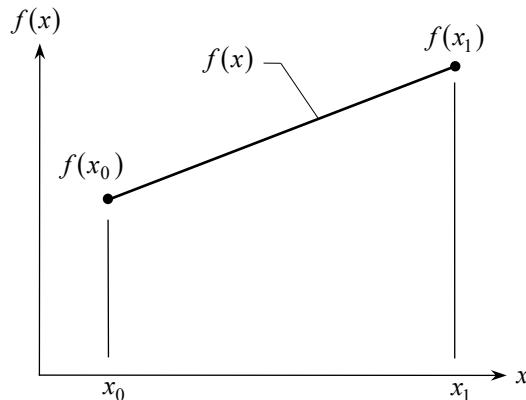


Figure 4.7 Linear interpolation between points x_0 and x_1 .

The straight line is represented by a linear function in the form

$$f(x) = ax + b \quad (4.17)$$

where a and b are constants and can be determined by using the data values at point x_0 and x_1 as follows

$$\text{at } x = x_0 : \quad f(x_0) = ax_0 + b \quad (4.18a)$$

$$\text{at } x = x_1 : \quad f(x_1) = ax_1 + b \quad (4.18b)$$

Equation (4.18b) is first subtracted from Eq. (4.18a) to yield

$$f(x_1) - f(x_0) = a(x_1 - x_0)$$

$$a = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Then, by substituting a into Eq. (4.18a) to give

$$b = f(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0} x_0$$

Thus, Eq. (4.17) becomes

$$f(x) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} x + f(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0} x_0$$

The above equation can be rewritten as

$$f(x) = \left(\frac{x_1 - x}{x_1 - x_0} \right) f(x_0) + \left(\frac{x_0 - x}{x_0 - x_1} \right) f(x_1) \quad (4.19)$$

or,

$$f(x) = L_0(x)f(x_0) + L_1(x)f(x_1) \quad (4.20)$$

where $L_0(x) = \frac{x_1 - x}{x_1 - x_0}$ and $L_1(x) = \frac{x_0 - x}{x_0 - x_1}$ (4.21)

The functions $L_0(x)$ and $L_1(x)$ are called the Lagrange interpolation functions. Their distributions are shown in Fig. 4.8.

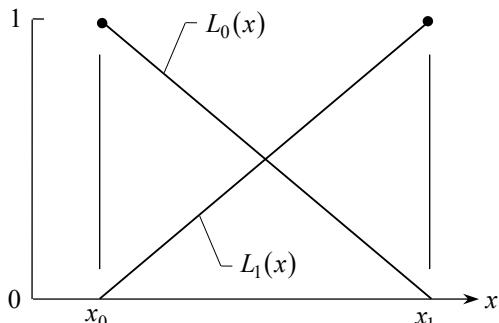


Figure 4.8 Linear Lagrange interpolation functions.

From Fig. 4.8,

$$L_0(x) = \begin{cases} 1 & ; x = x_0 \\ 0 & ; x = x_1 \end{cases} \quad (4.22a)$$

and

$$L_1(x) = \begin{cases} 0 & ; x = x_0 \\ 1 & ; x = x_1 \end{cases} \quad (4.22b)$$

Or, it can be concluded that

$$L_i(x) = \begin{cases} 1 & ; x = x_i \\ 0 & ; \text{at other points} \end{cases} \quad (4.23)$$

Example 4.4 Use linear Lagrange interpolation function to estimate the value of the Bessel function at $x = 3.2$. The values of the Bessel function at $x_0 = 2.0$ and $x_1 = 4.0$ are $f(x_0) = 0.2239$ and $f(x_1) = -0.3971$, respectively. Compare the estimated value with the exact solution in Table 4.1.

From Eq. (4.21), the linear Lagrange interpolation functions are

$$L_0(x) = \frac{x_1 - x}{x_1 - x_0} = \frac{4.0 - x}{4.0 - 2.0} \quad (4.24a)$$

$$L_1(x) = \frac{x_0 - x}{x_0 - x_1} = \frac{2.0 - x}{2.0 - 4.0} \quad (4.24b)$$

Then, the linear Lagrange interpolation is

$$f(x) = \left(\frac{4.0-x}{4.0-2.0} \right) f(x_0) + \left(\frac{2.0-x}{2.0-4.0} \right) f(x_1) \quad (4.25)$$

By substituting the values of $f(x_0)$ and $f(x_1)$ into Eq. (4.25), the estimated value at $x = 3.2$ is

$$\begin{aligned} f(x=3.2) &= \left(\frac{4.0-3.2}{4.0-2.0} \right)(0.2239) + \left(\frac{2.0-3.2}{2.0-4.0} \right)(-0.3971) \\ &= 0.08956 - 0.23826 \\ f(3.2) &= -0.1487 \end{aligned} \quad (4.26)$$

The result is identical to that obtained from the Newton's divided-difference method in Example 4.1 with the true error of 53.56%.

4.3.2 Quadratic interpolation

A quadratic function is used to fit a set of three data $f(x_0)$, $f(x_1)$, $f(x_2)$ at x_0 , x_1 , x_2 , respectively. The quadratic function is expressed in the form as shown in Fig. 4.9.

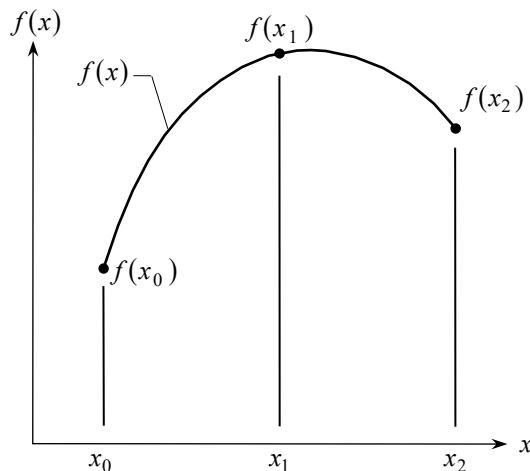


Figure 4.9 Quadratic interpolation passing through the three data points at x_0 , x_1 and x_2 .

Here

$$f(x) = ax^2 + bx + c \quad (4.27)$$

where a , b and c are constants and can be determined from the three data at x_0 , x_1 and x_2 as

$$\text{at } x = x_0 : \quad f(x_0) = ax_0^2 + bx_0 + c \quad (4.28a)$$

$$\text{at } x = x_1 : \quad f(x_1) = ax_1^2 + bx_1 + c \quad (4.28b)$$

$$\text{at } x = x_2 : \quad f(x_2) = ax_2^2 + bx_2 + c \quad (4.28c)$$

The procedure to determine the constants a , b and c is left as an exercise at the end of the chapter. After these constants are determined and substituted into Eq. (4.27), the quadratic Lagrange interpolation is obtained in the form

$$f(x) = L_0(x)f(x_0) + L_1(x)f(x_1) + L_2(x)f(x_2) \quad (4.29)$$

where

$$L_0(x) = \frac{(x_2 - x)(x_1 - x)}{(x_2 - x_0)(x_1 - x_0)} \quad (4.30a)$$

$$L_1(x) = \frac{(x_2 - x)(x_0 - x)}{(x_2 - x_1)(x_0 - x_1)} \quad (4.30b)$$

$$L_2(x) = \frac{(x_1 - x)(x_0 - x)}{(x_1 - x_2)(x_0 - x_2)} \quad (4.30c)$$

A typical Lagrange interpolation function (L_i) as shown in Eq. (4.30a-c) is equal to unity at x_i and zero at other points as shown in Fig. 4.10.

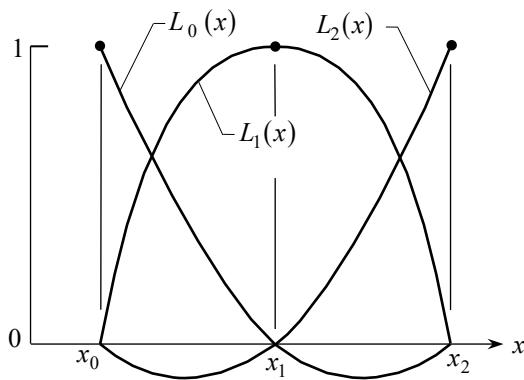


Figure 4.10 Quadratic Lagrange interpolation functions.

Example 4.5 Use the quadratic Lagrange interpolation to estimate the value of the Bessel function at $x = 3.2$. The three values of the Bessel function are given as $f(x_0 = 2.0) = 0.2239$, $f(x_1 = 3.0) = -0.2601$ and $f(x_2 = 4.0) = -0.3971$. Compare the estimated value with exact solution in Table 4.1.

From Eq. (4.30a-c), the quadratic Lagrange interpolation functions are

$$\begin{aligned} L_0(x) &= \frac{(x_2 - x)(x_1 - x)}{(x_2 - x_0)(x_1 - x_0)} = \frac{(4-x)(3-x)}{(4-2)(3-2)} = \frac{(4-x)(3-x)}{2} \\ L_1(x) &= \frac{(x_2 - x)(x_0 - x)}{(x_2 - x_1)(x_0 - x_1)} = \frac{(4-x)(2-x)}{(4-3)(2-3)} = -(4-x)(2-x) \\ L_2(x) &= \frac{(x_1 - x)(x_0 - x)}{(x_1 - x_2)(x_0 - x_2)} = \frac{(3-x)(2-x)}{(3-4)(2-4)} = \frac{(3-x)(2-x)}{2} \end{aligned}$$

Then, the quadratic Lagrange interpolation in Eq. (4.29) is

$$f(x) = \frac{(4-x)(3-x)}{2} f(x_0) - (4-x)(2-x) f(x_1) + \frac{(3-x)(2-x)}{2} f(x_2)$$

By substituting the data values of $f(x_0)$, $f(x_1)$ and $f(x_2)$ into the above equation, the estimated value at $x = 3.2$ is

$$\begin{aligned} f(x=3.2) &= \frac{(4-3.2)(3-3.2)}{2}(0.2239) - (4-3.2)(2-3.2)(-0.2601) \\ &\quad + \frac{(3-3.2)(2-3.2)}{2}(-0.3971) \\ &= -0.017912 - 0.249696 - 0.047652 \\ f(3.2) &= -0.3153 \end{aligned} \quad (4.31)$$

The estimated value is identical to that obtained from the Newton's divided-difference method in Example 4.2 with the true error of 1.53%.

4.3.3 Polynomial interpolation

By understanding the linear and quadratic Lagrange interpolations explained in the preceding sections, the n^{th} -order Lagrange interpolation as shown in Fig. 4.4 can be derived in the same manner. The general form of the n^{th} -order Lagrange interpolation can be written in the form

$$f(x) = \sum_{i=0}^n L_i(x) f(x_i) \quad (4.32)$$

where

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j} \quad (4.33)$$

The symbol \prod represents the multiplication operator. For example,

$$\prod_{k=1}^3 (k+1) = (1+1)(2+1)(3+1) = (2)(3)(4) = 24 \quad (4.34)$$

The use of the Lagrange interpolation function $L_i(x)$ in the form of Eq. (4.33) can be explained by using an example with three data points (i.e., $n = 2$). In this case, the function $L_1(x)$ is

$$L_1(x) = \prod_{\substack{j=0 \\ j \neq 1}}^2 \frac{x - x_j}{x_1 - x_j} = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

which is identical to that shown in Eq. (4.30b).

The Lagrange interpolation function in Eq. (4.33) is equal to unity at x_i and zero at the other points. Figure 4.11 shows the profile of a typical Lagrange interpolation function that has such properties.

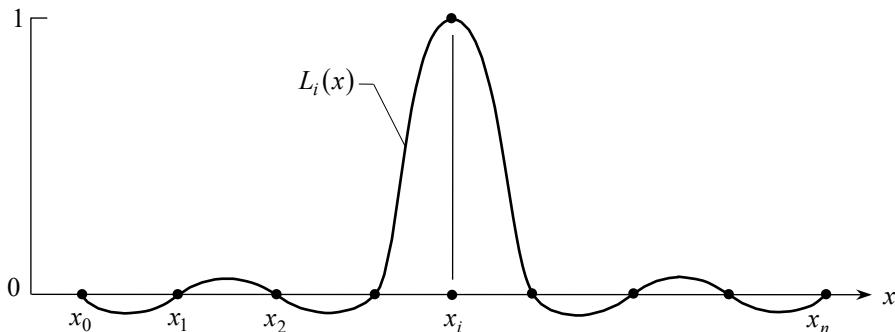


Figure 4.11 Profile of a typical Lagrange interpolation function.

From Examples 4.4 and 4.5, the linear and quadratic Lagrange interpolation functions can be derived easily from a few data points. For a general problem with many data points, the derivation of the Lagrange interpolation function is tedious and a computer program for determining them is needed. Figure 4.12 shows a computer program for determining the Lagrange interpolation functions for $n+1$ data points. The program, in addition, estimates the interpolated value from a given set of data points.

```
% Program LagPol
% Program for computing f(x) at a given x
% by using Lagrange interpolation.
% Read number of data set:
fid = fopen('input.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [2 n]);
x = a(1,:); x = x';
fx = a(2,:); fx = fx';
fclose(fid);
% Compute desired f(x) at the given x:
xx = input(... ...
'\nEnter x value to find f(x): ');
yy = 0.;
for i = 1:n
    al = 1.;
    for j = 1:n
        if j ~= i
            al = al*(xx-x(j))/(x(i)-x(j));
        end
    end
    yy = yy + al*fx(i);
end
fprintf('\nValue of f(x) at x = %8.6f', xx)
fprintf(' is %14.6e\n', yy)
```

Figure 4.12 Computer program for determining the Lagrange interpolation functions and the interpolated value from a given set of data.

Example 4.6 Use the computer program for determining the Lagrange interpolation functions as shown in Fig. 4.12 to estimate the value of the Bessel function at $x = 3.2$ from all data points in Table 4.1 except at $x = 3.2$. Compare the estimated value with that obtained from the Newton's divided-difference method in Example 4.3.

Figure 4.13 shows the input data needed for the computer program in Fig. 4.12. The data are from Table 4.1 except the one at $x = 3.2$. The computed value from the computer program is -0.3201 which is identical to that from the Newton's divided-difference method in Example 4.3.

It should be noted that some patterns of data points cannot be fitted well by using the higher-order polynomials. For example, the data points that are taken from the Runge's function which is represented by

$$f(x) = \frac{1}{1+25x^2} \quad (4.35)$$

```

10
2.0   .2239
2.2   .1104
2.4   .0025
2.6   -.0968
2.8   -.1850
3.0   -.2601
3.4   -.3643
3.6   -.3918
3.8   -.3992
4.0   -.3971
      3.2

Value of f(x) at x = .3200E+01 is  -.3201119E+00

```

Figure 4.13 Example of input data from Table 4.1 for using with the computer program in Fig. 4.12 and the computed value from the Lagrange interpolation method.

If the fourth-order Lagrange interpolation is used to fit the five data (at $x = -1, -0.5, 0, 0.5, 1$) taken from the Runge's function, its distribution is shown in Fig. 4.14. The figure shows that the distribution of the fourth-order Lagrange interpolation is quite different from the actual profile of the Runge's function. In this case, it is obvious that the Lagrange interpolation should not be used to represent the distribution of the Runge's function. In the next section, another interpolation method that can satisfactorily fit some special set of data or functions will be explained.

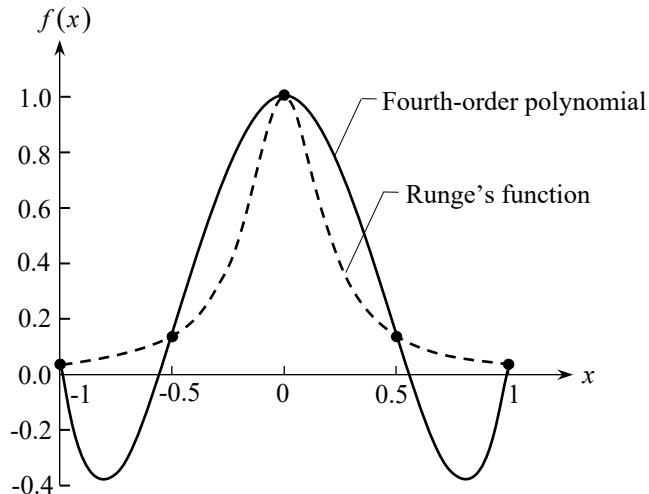


Figure 4.14 Comparison between the distribution fitted by the fourth-order polynomial and the actual profile of the Runge's function.

4.4 Spline Interpolations

The basic idea of the Newton's divided-difference and Lagrange interpolation methods is to create a polynomial function that passes through all data points. However, the distribution of such a single polynomial function may not realistically represent the true behavior of the problem. For example,

a set of eight data points for the air density measured from an experiment of a supersonic flow over an airfoil is shown in Fig. 4.15. Figure 4.15(a) shows a single polynomial function that passes through all eight data points. The distribution of the fitted polynomial is not realistic with up and down deviation between these data points. A more realistic distribution is obtained by using many functions to fit these data points as shown in Fig. 4.15(b). In this section, the spline interpolation method that can produce more realistic distributions of some particular data behavior is explained.

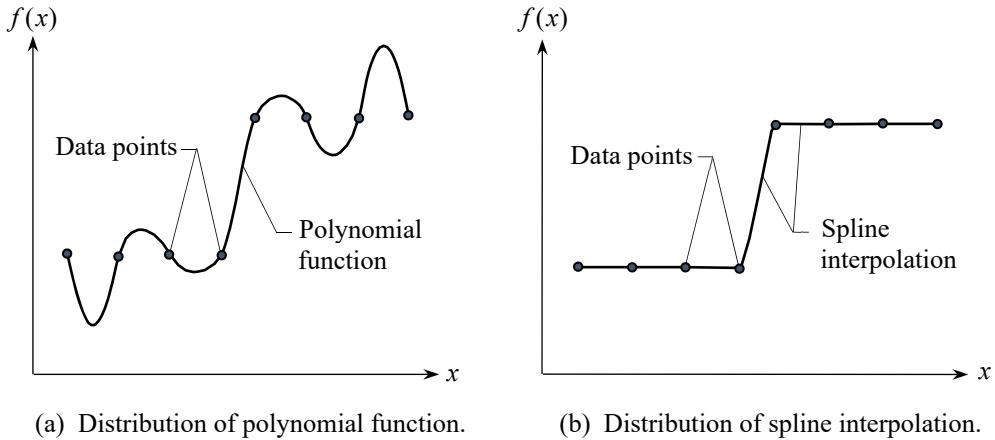


Figure 4.15 Distributions of a polynomial function and spline interpolation to fit a set of data distributed as a step function pattern.

4.4.1 Linear spline

The linear spline is the simplest spline function that uses a straight line to connect two data points. Figure 4.16 shows the three linear splines that are used to connect the four data points.

These three linear splines are

$$f_1(x) = f(x_0) + m_1(x - x_0) \quad ; \quad x_0 \leq x \leq x_1 \quad (4.36a)$$

$$f_2(x) = f(x_1) + m_2(x - x_1) \quad ; \quad x_1 \leq x \leq x_2 \quad (4.36b)$$

$$f_3(x) = f(x_2) + m_3(x - x_2) \quad ; \quad x_2 \leq x \leq x_3 \quad (4.36c)$$

where $m_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$ $i = 1, 2, 3$ (4.36d)

are their slopes. A computer program for creating the n linear splines from the given $n+1$ data points is easy to develop and is left as an exercise.

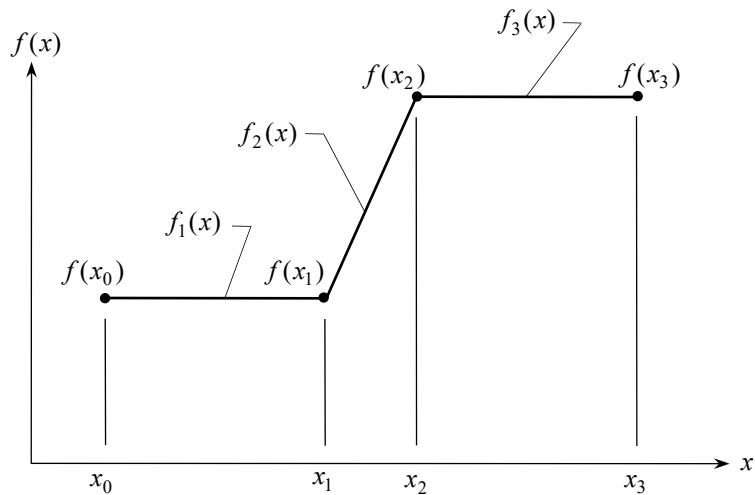


Figure 4.16 Three linear splines that connect four data points.

4.4.2 Quadratic Spline

The quadratic spline uses a quadratic function to represent distribution between the two data points. Figure 4.17 shows the three quadratic splines that pass through the four data points. These three quadratic splines are

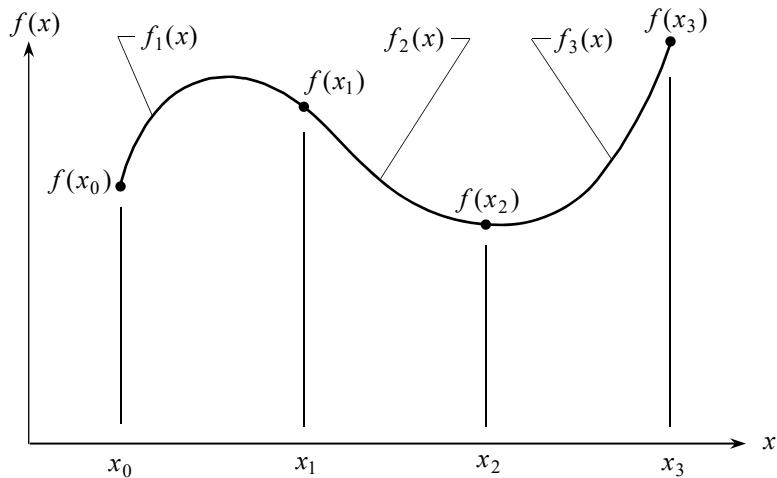


Figure 4.17 Three quadratic splines that connect four data points.

$$f_1(x) = a_1x^2 + b_1x + c_1 \quad ; \quad x_0 \leq x \leq x_1 \quad (4.37a)$$

$$f_2(x) = a_2x^2 + b_2x + c_2 \quad ; \quad x_1 \leq x \leq x_2 \quad (4.37b)$$

$$f_3(x) = a_3x^2 + b_3x + c_3 \quad ; \quad x_2 \leq x \leq x_3 \quad (4.37c)$$

where a_i , b_i , c_i , $i=1, 2, 3$ are unknown constants. These unknowns are determined by using the conditions as follows.

(a) At any internal point, the values of the two quadratic splines connected at that point must be equal. For example, the conditions at point x_1 in Fig. 4.17 are

$$f_1(x_1) = a_1x_1^2 + b_1x_1 + c_1 = f(x_1) \quad (4.38a)$$

and

$$f_2(x_1) = a_2x_1^2 + b_2x_1 + c_2 = f(x_1) \quad (4.38b)$$

Thus, the two internal points x_1 and x_2 in Fig. 4.17 yield 4 conditions.

(b) The first quadratic spline must pass through the first data point x_0 ,

$$f_1(x_0) = a_1x_0^2 + b_1x_0 + c_1 = f(x_0) \quad (4.39a)$$

and the third quadratic spline must pass through the last data point x_3 ,

$$f_3(x_3) = a_3x_3^2 + b_3x_3 + c_3 = f(x_3) \quad (4.39b)$$

so that two additional conditions are produced.

(c) At any internal point, the slopes of the two quadratic splines connected at that point must be equal. For example, the condition at point x_1 is

$$f'_1(x_1) = f'_2(x_1) \quad (4.40a)$$

$$2a_1x_1 + b_1 = 2a_2x_1 + b_2 \quad (4.40b)$$

The requirement for equal slopes yields two conditions for the two internal points in Fig. 4.17.

The total number of conditions for the four data points using the three quadratic splines is $4 + 2 + 2 = 8$. Since there are 9 unknowns, the coefficient a_1 may be assigned to be zero which means $f_1(x)$ is a linear spline. The eight unknowns can then be solved from the system of eight equations

$$\left[\begin{array}{|c|} \hline (8 \times 8) \\ \hline \end{array} \right] \begin{Bmatrix} b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} (8 \times 1) \\ \vdots \end{Bmatrix} \quad (4.41)$$

For a general problem with $n + 1$ data points, the total of n quadratic splines of $f_1(x)$, $f_2(x)$, ..., $f_n(x)$ is needed. Since each quadratic spline has 3 unknowns, so the total number of unknowns is $3n$. However, the number of equations that can be created are $2(n-1) + 2 + (n-1) = 3n - 1$ which is one condition fewer than the number of unknowns. If the first quadratic spline is assumed to be linear by setting its coefficient a_1 as zero, then the number of unknowns reduces to $3n - 1$ so that they can be solved from $3n - 1$ equations.

A computer program for determining the quadratic splines can be developed to create the system of Eq. (4.41) from the conditions as shown in Eqs. (4.38) - (4.40). This system of equations can be solved for the unknown coefficients by a procedure for solving a system of equations explained in Chapter 3. The quadratic splines are then obtained after substituting the computed coefficients into Eqs. (4.37a-c).

4.4.3 Cubic spline

The procedure for creating a cubic spline is similar to the linear and quadratic splines as explained in the preceding sections. Distribution of the quadratic spline is either in a convex or concave shape. The cubic spline is widely used because its distribution can be in both the convex and concave shapes between two data points.

If three cubic splines are to be created from the four data points in Fig. 4.17, the three cubic functions connecting the three pairs of data points are

$$f_1(x) = a_1x^3 + b_1x^2 + c_1x + d_1 \quad ; \quad x_0 \leq x \leq x_1 \quad (4.42a)$$

$$f_2(x) = a_2x^3 + b_2x^2 + c_2x + d_2 \quad ; \quad x_1 \leq x \leq x_2 \quad (4.42b)$$

$$f_3(x) = a_3x^3 + b_3x^2 + c_3x + d_3 \quad ; \quad x_2 \leq x \leq x_3 \quad (4.42c)$$

where a_i , b_i , c_i , d_i , $i=1, 2, 3$ are unknown coefficients. The twelve unknown coefficients of the three cubic functions can be determined from the following conditions.

(a) At any internal point, the values of the two cubic splines connected at that point must be equal. For example, the conditions at point x_1 in Fig. 4.17 are

$$f_1(x_1) = a_1x_1^3 + b_1x_1^2 + c_1x_1 + d_1 = f(x_1) \quad (4.43a)$$

$$\text{and} \quad f_2(x_1) = a_2x_1^3 + b_2x_1^2 + c_2x_1 + d_2 = f(x_1) \quad (4.43b)$$

Thus, the two internal points x_1 and x_2 in Fig. 4.17 yield four conditions.

(b) The first cubic spline must pass through the first data point x_0 ,

$$f_1(x_0) = a_1x_0^3 + b_1x_0^2 + c_1x_0 + d_1 = f(x_0) \quad (4.44a)$$

and the last cubic spline must pass through the last data point, x_3 ,

$$f_3(x_3) = a_3x_3^3 + b_3x_3^2 + c_3x_3 + d_3 = f(x_3) \quad (4.44b)$$

so that two additional conditions are produced.

(c) At any internal point, the slopes of the two cubic splines connected at that point must be equal. For example, the condition at point x_1 is

$$f'_1(x_1) = f'_2(x_1) \quad (4.45a)$$

$$3a_1x_1^2 + 2b_1x_1 + c_1 = 3a_2x_1^2 + 2b_2x_1 + c_2 \quad (4.45b)$$

The requirement for equal slopes yields two conditions for the two internal points in Fig. 4.17.

(d) At any internal point, the second derivatives of the two cubic splines connected at that point must be equal. For example, the condition at point x_1 is

$$f''_1(x_1) = f''_2(x_1) \quad (4.46a)$$

$$6a_1x_1 + 2b_1 = 6a_2x_1 + 2b_2 \quad (4.46b)$$

The requirement for equal second-derivatives yields two conditions for the two internal points in Fig. 4.17.

(e) The conditions of zero second-derivative for the cubic splines at the first and last data points may be assigned to produce two more conditions of

$$f''_1(x_0) = f''_3(x_3) = 0 \quad (4.46c)$$

From the set of four data points connected by the three cubic splines in Fig. 4.17, there is the total of 12 unknown coefficients. These unknown coefficients are determined from the $4 + 2 + 2 + 2 + 2 = 12$ conditions from Eqs. (4.43)-(4.46). The twelve equations can be written in matrix form as

$$(12 \times 12) \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ \vdots \\ a_3 \\ b_3 \\ c_3 \\ d_3 \end{bmatrix} = \begin{bmatrix} (12 \times 1) \end{bmatrix} \quad (4.47)$$

For a set of $n+1$ data points to be fitted by n cubic splines $f_1(x), f_2(x), \dots, f_n(x)$, there are $4n$ unknown coefficients to be determined. These unknown coefficients are solved from a set of $2(n-1) + 2 + (n-1) + (n-1) + 2 = 4n$ equations according to the conditions explained in Eqs. (4.43) - (4.46).

Solving a set of $4n$ equations to produce n cubic splines may not be simple, especially for the case of a large value of n . An alternative approach explained below solves only $n-1$ equations in order to produce n cubic splines. The idea behind such approach is to assume a linear distribution for the second derivative of the spline function $f_i(x)$. Such linear distribution can be written in form of the Lagrange interpolation as

$$f''_i(x) = f''(x_{i-1}) \frac{x - x_i}{x_{i-1} - x_i} + f''(x_i) \frac{x - x_{i-1}}{x_i - x_{i-1}} \quad (4.48)$$

where $i = 1, 2, \dots, n$. The spline function $f_i(x)$ can then be obtained by performing integration twice. The integrations lead to two integrating constants which can be determined from the two conditions of

$$\text{at point } x = x_{i-1} : f_i(x_{i-1}) = f(x_{i-1}) \quad (4.49a)$$

$$\text{and at point } x = x_i : f_i(x_i) = f(x_i) \quad (4.49b)$$

Once the two integrating constants are determined and substituted back, the final spline function is obtained in the form,

$$\begin{aligned} f_i(x) &= \frac{f''(x_{i-1})}{6(x_i - x_{i-1})}(x_i - x)^3 + \frac{f''(x_i)}{6(x_i - x_{i-1})}(x - x_{i-1})^3 \\ &\quad + \left[\frac{f(x_{i-1})}{(x_i - x_{i-1})} - \frac{(x_i - x_{i-1})f''(x_{i-1})}{6} \right](x_i - x) \\ &\quad + \left[\frac{f(x_i)}{(x_i - x_{i-1})} - \frac{(x_i - x_{i-1})f''(x_i)}{6} \right](x - x_{i-1}) \end{aligned} \quad (4.50)$$

The right-hand side of the derived spline function in Eq. (4.50) consists of the two unknown second-derivatives at x_{i-1} and x_i . It is noted that the first derivative of the two spline functions connected at an interior point must be equal,

$$f'_{i-1}(x_i) = f'_i(x_i) \quad (4.51)$$

The first derivatives of the spline functions $f_{i-1}(x)$ and $f_i(x)$ can be derived from Eq. (4.50). These derivatives are then required to be equal according to Eq. (4.51) which leads to

$$\begin{aligned} &(x_i - x_{i-1})f''(x_{i-1}) + 2(x_{i+1} - x_{i-1})f''(x_i) + (x_{i+1} - x_i)f''(x_{i+1}) \\ &= \frac{6}{x_{i+1} - x_i}(f(x_{i+1}) - f(x_i)) + \frac{6}{x_i - x_{i-1}}(f(x_{i-1}) - f(x_i)) \end{aligned} \quad (4.52)$$

Equation (4.52) consists of the three unknowns which are the second derivatives of the spline functions at x_{i-1} , x_i and x_{i+1} , where $i = 1, 2, \dots, n$. If the second derivatives of the spline functions for the first and last data points are assumed to be zero,

$$f''(x_0) = f''(x_n) = 0 \quad (4.53)$$

then, Eq. (4.52) yields a set of simultaneous equations with $n-1$ unknowns of $f''(x_1)$, $f''(x_2)$, \dots , $f''(x_{n-1})$ in the form

$$\begin{bmatrix} x & x & & & & \\ x & x & x & & & \\ & x & x & x & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \ddots \\ & & & & x & x & x \\ & & & & x & x & \end{bmatrix} \begin{Bmatrix} f''(x_1) \\ f''(x_2) \\ f''(x_3) \\ \vdots \\ \vdots \\ f''(x_{n-2}) \\ f''(x_{n-1}) \end{Bmatrix} = \begin{Bmatrix} x \\ x \\ x \\ \vdots \\ \vdots \\ x \\ x \end{Bmatrix} \quad (4.54)$$

where the symbol x represents the non-zero coefficient in the matrix. The square matrix on left-hand side of Eq. (4.54) is a tridiagonal matrix because all elements in the matrix are zero except those along the three main diagonals. Such system of equations can be solved conveniently by using the methods explained in Chapter 3.

After solving the system of equations in Eq. (4.54), the second derivatives are substituted into Eq. (4.50) to yield the desired cubic spline functions between x_{i-1} and x_i . The following example demonstrates the performance of the cubic spline functions for fitting a set of special data as compared to that from the high-order polynomial interpolation.

Example 4.7 Develop a computer program that employs the cubic spline interpolation as explained in Eqs. (4.48) - (4.54) to fit the data given in the table below. Compare the result with that obtained from the use of high-order Lagrange interpolation. It is noted that the result from the use of high-order Lagrange interpolation can be obtained from the computer program in Fig. 4.12.

x	0.00	0.10	0.30	0.44	0.56	0.60	0.70	0.85	1.00
$f(x)$	0.1	0.1	0.1	0.1	0.5	0.5	0.5	0.5	0.5

Figure 4.18 shows the computer program for cubic spline interpolation. The procedure in the program starts from reading the data and the location needed for the interpolated result. Next, the program forms the system of equations as shown in Eq. (4.54) and solves such system of equations to obtain the second derivatives for all data points. Finally, the values of the computed second derivatives are substituted into Eq. (4.50) to yield the cubic spline interpolation.

Figure 4.19 shows the comparison between the distributions obtained from the cubic spline interpolation and high-order Lagrange polynomial. The figure shows that the cubic spline interpolation provides realistic distribution while the high-order Lagrange polynomial yields oscillated distribution. The example also suggests that users should understand nature of the data before applying an appropriate interpolating method.

```
% Program CubSpls
% A cubic spline interpolating program.
% Read number of data set, x and fx data.
fid = fopen('inputCS.dat', 'r');
n = fscanf(fid, '%f', 1);
p = fscanf(fid, '%f', [2 n]);
x = p(1,:); x = x';
fx = p(2,:); fx = fx';
fclose(fid);
% Form up triadiagonal system of n eqs:
for i = 2:n-1
    a(i)=x(i) - x(i-1);
    b(i)=2.0*(x(i+1) - x(i-1));
    c(i)=x(i+1) - x(i);
    d(i)=6.0*(fx(i+1)-fx(i))/(x(i+1)-x(i)) ...
        + 6.0*(fx(i-1)-fx(i))/(x(i)-x(i-1));
end
b(1) = 1.0; c(1) = 0.0; d(1) = 0.0;
a(n) = 0.0; b(n) = 1.0; d(n) = 0.0;
% Solve the tridiagonal system of n eqs.
% for the second derivatives, return the
% computed solution in e():
for i = 2:n
    a(i) = a(i)/b(i-1);
    b(i) = b(i) - a(i)*c(i-1);
    end
    for i = 2:n
        d(i) = d(i) - a(i)*d(i-1);
    end
    e(n) = d(n)/b(n);
    for i = n-1:-1:1
        e(n) = (d(i) - c(i)*e(i+1))/b(i);
    end
    xx = input( ...
        '\nEnter x value to find f(x): ');
    for i = 2:n
        if (xx >= x(i-1)) && (xx <= x(i)))
            d1 = x(i) - xx;
            d2 = xx - x(i-1);
            dd = x(i) - x(i-1);
            t1 = e(i-1)*d1*d1*d1/(6.0*dd);
            t2 = e(i)*d2*d2*d2/(6.0*dd);
            t3 = (fx(i-1)/dd-e(i-1)*dd/6.0)*d1;
            t4 = (fx(i)/dd-e(i)*dd/6.0)*d2;
            ff = t1 + t2 + t3 + t4;
        end
    end
    fprintf('\nValue of f(x) at x = %8.6f', xx)
    fprintf(' is %14.6e\n', ff)
```

Figure 4.18 Computer program for cubic spline interpolation method.

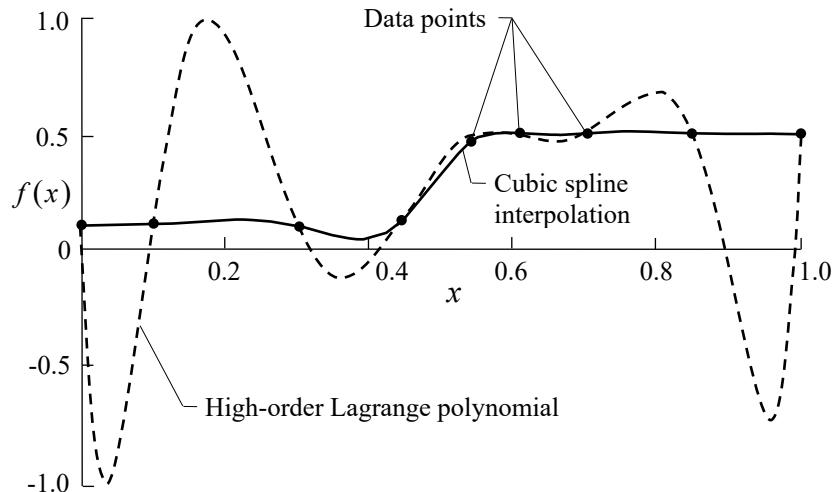


Figure 4.19 Comparison between the distributions obtained from using the cubic spline interpolation and higher-order Lagrange polynomial.

4.5 MATLAB Functions for Interpolations

Two built-in MATLAB functions that can be used for interpolation are explained herein. The two functions are `spline` and `interp1`. The `spline` function is based on the cubic spline interpolation method explained in the preceding section. The `interp1` function allows the user to select a method for the interpolation.

The `spline` function has the syntax as follow,

$$\text{yy} = \text{spline}(x, y, xx)$$

where `x` and `y` are the locations and values of the data
`xx` is the location to perform interpolation
`yy` is the interpolated value at the location `xx`

Example 4.8 Employ the `spline` function to fit the Runge's function,

$$f(x) = \frac{1}{1+25x^2} \quad (4.35)$$

by using nine data at $x = -1.0, -0.75, -0.5, -0.25, 0.0, 0.25, 0.5, 0.75$ and 1.0 . Plot to compare the computed distribution with the actual distribution of the Runge's function.

The values at the specified nine locations can be generated by using the

```
>> x = [-1:0.25:1];
>> y = 1./(1+25*x.^2);
```

Then, the `spline` function is employed to determine the values of `yy` at `xx` starting from -1 to 1 with the interval of 0.125 as follows

```
>> xx = [-1:0.125:1];
>> yy = spline(x,y,xx);
```

Figure 4.20 shows the comparison of the distribution from the values by the `spline` function (dashed line) and the actual distribution of the Runge's function (solid line). The comparison shows that the distribution from the values by the `spline` function agrees very well with the Runge's function.

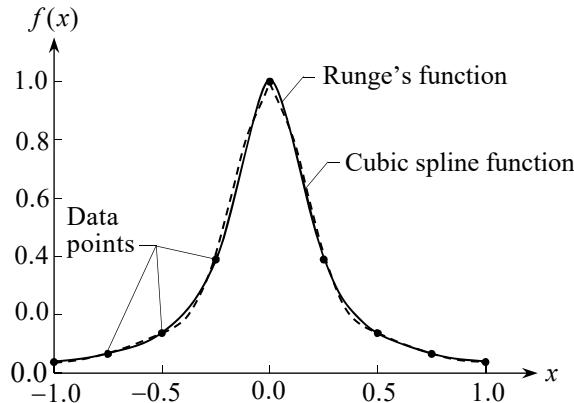


Figure 4.20 Comparison between the distribution from the values by the `spline` function and the actual distribution of the Runge's function.

The second useful function for interpolation in MATLAB is `interp1`. The syntax of this function is

```
yy = interp1(x, y, xx, 'method')
```

where `x` and `y` are the locations and values of the data

`xx` is the location to perform interpolation

`yy` is the interpolated value at the location `xx`

'`method`' is the option for interpolation:

- 'nearest' The option interpolates the value at the location `xx` by using the values from the nearest locations

- 'linear' The option uses linear interpolation to determine the value at the location `xx`

- 'spline' The option uses cubic spline interpolation to determine the value at the location `xx`

The default '`method`' is '`linear`' if one of the three options above is not selected.

4.6 Extrapolation

All methods in the preceding sections explain the interpolation procedures. These methods create a function $f(x)$ passing through all of the data at $x_0, x_1, x_2, \dots, x_n$. The function is then used to interpolate a value at the desired location between x_0 and x_n . Accuracy of the interpolated value depends on the number of the given data points and its location. The accuracy of the interpolated value also increases if the desired location is closed to a given data point.

The basic idea of the extrapolation is to use the function created from an interpolation method to estimate the value outside the range of the given data points. Figure 4.21 shows an example of the interpolation and extrapolation from the three data points at x_0, x_1 and x_2 . For interpolation, a quadratic function can be created to estimate the value within the given range. The figure shows that the quadratic function can provide accurate interpolated value between x_0 and x_2 . The same quadratic function, however, may yield an extrapolated value with a large error at a location outside the range of x_0 and x_2 . The figure suggests that extrapolation must be performed with care, especially if the number of data points is a few. The following two examples show that the accuracy of the extrapolated value can be improved by increasing the number of data points.

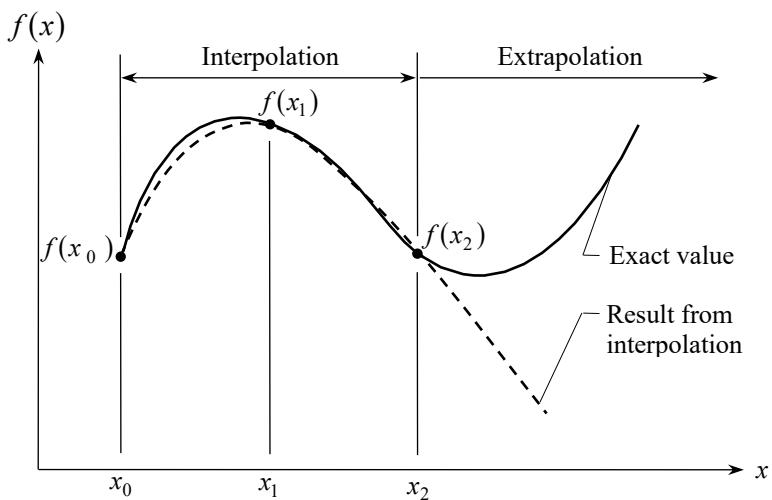


Figure 4.21 Comparison between interpolation and extrapolation.

Example 4.9 The values of the Bessel's function at three locations are given by $f(x_0 = 2.0) = 0.2239$, $f(x_1 = 2.6) = -0.0968$ and $f(x_2 = 3.2) = -0.3202$. Determine the value of the Bessel's function at point $x = 4.0$ by performing extrapolation.

From the given three data points, the computer program for the Newton's divided-difference method as shown in Fig. 4.5 can be used to extrapolate the value of the Bessel's function at $x = 4.0$. The extrapolated value is $f(x = 4.0) = -0.4667$. The extrapolated value has the true error as compared to the exact solution in Table 4.1 as

$$\varepsilon = \frac{-0.3971 + 0.4667}{-0.3971} \times 100\% = -17.53\% \quad (4.55)$$

The error from the extrapolated value in Eq. (4.55) is relatively large. The same extrapolation is performed again but by using more data points as shown in the following example.

Example 4.10 The values of the Bessel's function at five locations are given by $f(x_0 = 2.0) = 0.2239$, $f(x_1 = 2.4) = 0.0025$, $f(x_2 = 2.8) = -0.1850$, $f(x_3 = 3.2) = -0.3202$ and $f(x_4 = 3.6) = -0.3918$. Determine the value of the Bessel's function at point $x = 4.0$ by performing extrapolation.

From the values of the Bessel's function at five locations, the computer program in Fig. 4.5 can be used again to determine the value of the Bessel's function at point $x = 4$. The extrapolated value is $f(x = 4.0) = -0.3956$. The extrapolated value has the true error as compared to the exact solution in Table 4.1 as

$$\varepsilon = \frac{-0.3971 + 0.3956}{-0.3971} \times 100\% = -0.38\% \quad (4.56)$$

The error from this later Example 4.10 is quite small as compared to that from Example 4.9. These two examples demonstrate that the error of the extrapolated value can be decreased by increasing the number of the data points.

4.7 Closure

In this chapter, the interpolation and extrapolation methods are presented. Both methods estimate values at locations from a set of data points. The interpolation methods presented herein are the Newton's divided differences, Lagrange polynomial and spline interpolations. The Newton's divided differences and Lagrange interpolations create the n^{th} -order polynomial that pass through $n+1$ data points. Linear, quadratic and general n^{th} -order polynomials are derived and explained in details with examples. If the given set of data points behaves properly, accurate interpolated values can be obtained by using high-order polynomials.

For some special sets of data points, high-order polynomials may yield oscillated distributions leading to inaccurate interpolated values. In this case, the spline interpolation methods may be used. The most popular spline interpolation method is to employ a cubic function to represent the data distribution between two data points. Derivation of the cubic spline interpolation is explained and presented in details with examples. For these interpolation methods, their corresponding computer programs are also developed so that the interpolated values from a large set of data points can be obtained conveniently.

Extrapolation to estimate values outside the given range of data by using the Newton's divided differences and Lagrange polynomials is presented at the end of the chapter. The methods and associate examples suggest that the extrapolation must be performed with care. The extrapolated values may be inaccurate from a set of few data points or their locations are far away from the given data locations. It is noted that some of the interpolation methods, such as the use of the Lagrange polynomials, are the basis for studying the finite difference and finite element methods in the later chapters.

Exercises

1. Relation between the applied force P and displacement u of the three-bar truss structure in Fig. P4.1 is nonlinear according to the data below

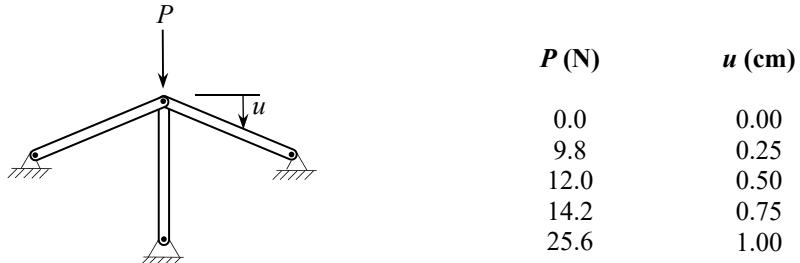


Figure P4.1

- (a) Plot the data to show the nonlinear relation between the force and displacement.
 - (b) Derive the Lagrange polynomial for the given data points.
 - (c) Plot the Lagrange polynomial obtained in (b) and compare with the data in (a).
 - (d) Use the derived Lagrange polynomial to estimate the forces at the displacements of 0.17, 0.62 and 1.25 cm.
2. From an experiment, the heat conduction coefficient k of an aluminum material varies with the temperature T as shown in the table. Derive the Lagrange interpolation polynomial to estimate values of the heat conduction coefficient at 50°C, 250 °C and 500 °C.

T (°C)	-100	0	100	200	300	400
k (W/m·°C)	215	202	206	215	228	249

3. Values of the gravitational acceleration g depend on the altitude y as shown in table. Use the Newton's divided difference method to estimate the value of gravitational accelerations at the altitudes of 5,000 m, 42,000 m and 90,000 m. Plot the distribution of the derived interpolation function together with the given data.

y (m)	0	20,000	40,000	60,000	80,000
g (m/sec ²)	9.8100	9.7487	9.6879	9.6278	9.5682

4. The air density ρ depends on the temperature T as shown in the table. Employ the computer program of the Newton's divided-difference method in Fig. 4.5 to estimate values of the air density at 250 K, 800 K and 3,000 K. Then, modify the computer program to determine the air density at the temperature at every 10 K from 100 K to 2,500 K. Plot the computed air density distribution that varies with the temperature.

T (K)	100	200	300	500	700	1,000	1,500	2,000	2,500
ρ (kg/m ³)	3.6010	1.7684	1.1774	0.7048	0.5030	0.3524	0.2355	0.1762	0.1394

5. Show that the Lagrange interpolation functions in Eqs. (4.20) and (4.29) are equivalent to the interpolation functions obtained from Newton's divided-difference method in Eqs. (4.1) and (4.6).
6. The set of data in the table below represents the displacement u of a spring from the applied force F . Employ the computer program of the Newton's divided differences or Lagrange interpolation method as shown in Figs. 4.5 and 4.12 to estimate the force needed to displace the spring to 0.3 m and 0.4 m. Then, plot the function obtained from the computer program along with the given data points.

u (m)	0.107	0.172	0.238	0.351	0.388	0.417	0.432	0.441
F (kN)	100	200	300	400	500	600	700	800

7. Temperature drop from wind chill depends on the wind speed. A set of the temperature data that varies with the wind speed is shown in the table. Use the Newton's divided-difference method to estimate the temperature at the wind speed of 35 km/hr. From the given set of data, is it possible for the temperature to drop lower than -50 °C. If it is possible, find the wind speed that causes such temperature.

Wind speed (km/hr)	0	10	20	30	40	50
Temperature (°C)	-12	-23	-31	-36	-38	-39

8. Show detailed procedure for deriving the Lagrange quadratic interpolation functions in Eqs. (4.29) - (4.30).
9. Develop a computer program for the linear interpolation as explained in section 4.4.1. The program should be able to solve any problem with at least 100 data points. Verify the computer program with the set of data in Example 4.7.
10. Use the cubic spline interpolation method explained in section 4.4.3 to derive a set of functions that fit the data as shown in the table. Explain the computational steps in details and determine the value of $f(x = 5)$. Then, compare the computed value with that obtained from the computer program in Fig. 4.18.

x	1	4	6	9	10
$f(x)$	4	9	15	7	3

11. Develop a computer program for the quadratic spline interpolation as explained in section 4.4.2. The program should be able to solve any problem with at least 100 data points. Verify the computer program with the set of data in Example 4.7.

12. From the data in the table as shown below, derive the interpolation functions by using

x	$f(x)$	
2	9.5	(a) the Newton's differences method
4	8.0	(b) the Lagrange polynomial method
6	10.5	(c) the cubic spline method
8	39.5	
10	72.0	Plot the functions obtained from (a)-(c) together with the data points. Then, determine values of these functions at $x = 7$.

13. Fit the data in Problem 12 again but by using the quadratic spline interpolation as explained in section 4.4.2. Show the computational steps in details.

14. Solve Problem 1 again but by using the cubic spline interpolation method as explained in section 4.4.3. Verify the solution by using the computer program in Fig. 4.18. Plot the distributions of the interpolations obtained from these two problems. Then, provide comments on the advantages and disadvantages of each method.

15. From the data in the table as shown below, derive the interpolation functions by using

x	$f(x)$	
0	-7	(a) the Newton's differences method
1	-3	(b) the Lagrange polynomial method
2	6	(c) the cubic spline method
3	25	Determine the values from each function at $x = 2.5$ and 4.5 . Then,
4	62	compare the computed values with those obtained from the computer
5	129	programs in Figs. 4.5, 4.12 and 4.18.

16. Use the computer programs for the Lagrange polynomial and cubic spline methods as shown in Figs. 4.12 and 4.18 to determine the interpolation functions from the set of data points in Problem 15. Plot the functions obtained and discuss the advantages and disadvantages of each method.

17. Solve Problem 6 again but by using the cubic spline method as explained in section 4.4.3. Plot the distribution of the derived functions and compare them with that obtained from Problem 6.

18. The Runge's function is given by $f(x) = 1/(1+25x^2)$ for $-1 \leq x \leq 1$ as shown in Fig. 4.14. Create a set of data points from this function in the given range with an increment of $\Delta x = 0.2$. Then, fit the set of data with the tenth-order Lagrange polynomial function. Plot and compare the derived polynomial function with the Runge's function. Discuss the accuracy of the derived Lagrange polynomial function.

19. Solve Problem 18 again but by using the computer program for the cubic spline interpolation in Fig. 4.18. Plot to compare the result with the function obtained from Problem 18.

20. The Runge's function is given by $f(x) = 1/(1+25x^2)$ for $-5 \leq x \leq 5$. Create a set of data points from this function in the given range with $\Delta x = 1$. Then, derive the interpolation function by using the Newton's divided-difference method. Plot the distribution of the derived function and compare it with the Runge's function.
21. Solve Problem 20 again but by using the computer program for the cubic spline interpolation in Fig. 4.18. Plot to compare the result with the function obtained from Problem 20.
22. The Runge's function is given by $f(x) = 1/(1+25x^2)$ for $-2 \leq x \leq 2$. Create a set of data points from this function in the given range with $\Delta x = 1$ and then derive the interpolation function by using the Lagrange polynomial and cubic spline interpolation methods. Plot the derived functions to compare with the Runge's function. Discuss the accuracy of these derived functions for representing the Runge's function.
23. The data in the table below are generated from the function $f(x) = 100/x^2$. Use an extrapolation method to estimate the value of the function at $x = 5.7$. Compare the estimated value with the exact solution. Suggest ways to improve the solution accuracy from the extrapolation.

x	1	2	3	4	5
$f(x)$	100.000	25.000	11.111	6.250	4.000

24. Data in the table below represent the distribution of the air density ρ in front of an airfoil which moves at a supersonic speed as shown in Fig. 4.1. Use the computer program in Fig. 4.18 to derive the cubic spline interpolation function. Plot to compare such function with the given data and discuss the result. Provide comments if the Lagrange polynomial interpolation method is used to fit the same set of data.

x	ρ	x	ρ
1.000	4.2	1.200	15.2
1.020	4.8	1.204	18.7
1.040	5.1	1.208	23.5
1.060	5.2	1.212	28.9
1.080	5.3	1.216	34.0
1.100	5.5	1.220	38.3
1.120	5.8	1.228	42.7
1.140	6.1	1.236	45.3
1.160	6.5	1.244	46.2
1.180	7.4	1.250	46.4
1.189	9.1	1.300	46.4
1.196	12.9	1.400	46.4

25. From the table shown in Problem 4, use the MATLAB function `interp1` with `spline` option to interpolate the air density values from 100 K to 2,500 K at every 10 K. Compare the result obtained from MATLAB with those from Problem 4.
26. Solve Problem 6 again but by using the function `spline` in MATLAB. Compare the result with that obtained from the Lagrange polynomial interpolation.
27. From the table in Problem 10, use functions `spline` and `interp1` in MATLAB with `linear` option to interpolate the value of function at $x = 5$. Compare the result with that obtained from Problem 10.
28. From an experiment of air flow in a pipe, the average air velocity \bar{u} depends the distance y from the pipe surface as shown in the table below.

\bar{u}	0.318	0.300	0.264	0.228	0.221	0.179	0.152	0.140
y	0.0075	0.0071	0.0061	0.0055	0.0051	0.0041	0.0034	0.0030

Use the MATLAB function `spline` to determine the average velocities \bar{u} from $y = 0.0030$ to 0.0075 with the increment at every 0.0001. Plot and compare the computed results with the data in the table.

29. Performance curve of a pump is obtained from a set of data between the head and flow rate. The table below shows a set of data for the head (H) and flow rate (Q).

H (m)	45.1	42.7	39.6	35.1	30.5	22.9	15.2
Q (m³/hr)	0	180	270	360	450	540	630

Use a function in MATLAB to determine value of the head at the flow rate of 400 m³/hr.

Chapter

5

Least-Squares Regression

5.1 Introduction

Most of the material properties used in engineering design and analysis were obtained from experiments. For example, the material elasticity curve is needed in the stress analysis for designing a new product. Such curve is obtained from a function that best fits the experimental data. In chapter 4, procedures for deriving an interpolating function that matches the given data points exactly are presented. Use of the interpolating function is suitable for few data points that vary smoothly. For a large set of data points, especially when the data deviate considerably, an interpolating function may not provide accurate representation of the overall data behavior.

Derivation of an approximate function that best fits a given set of data points by using the least-squares regression is explained in this chapter. The method minimizes the sum of the squares of the differences between the data values and the values of the approximate function. To ease understanding of the method, a set of data for the wind velocities measured at different elevations of a building is shown in Table 5.1. The data indicate that the measured wind velocity increases with the building elevation.

Table 5.1 The measured wind velocities at different elevations of the building.

Building elevation, x (m)	Wind velocity, y (m/sec)
10	2.2
15	4.6
20	4.2
25	7.0
30	6.6
35	9.2

The data of the wind velocities at different elevations of the building as shown in Table 5.1 are plotted in Fig. 5.1. Due to the deviation of the measured wind velocities at different elevations, the function obtained from the interpolation method can not sufficiently represent the realistic behavior of the phenomenon as shown in the figure. Figure 5.1 also shows two approximate functions fitted by using eye-ball. These approximate functions do not represent the best fitted function for such set of data. The least-squares regression methods that are explained in this chapter provide the best fitted function for any set of data.

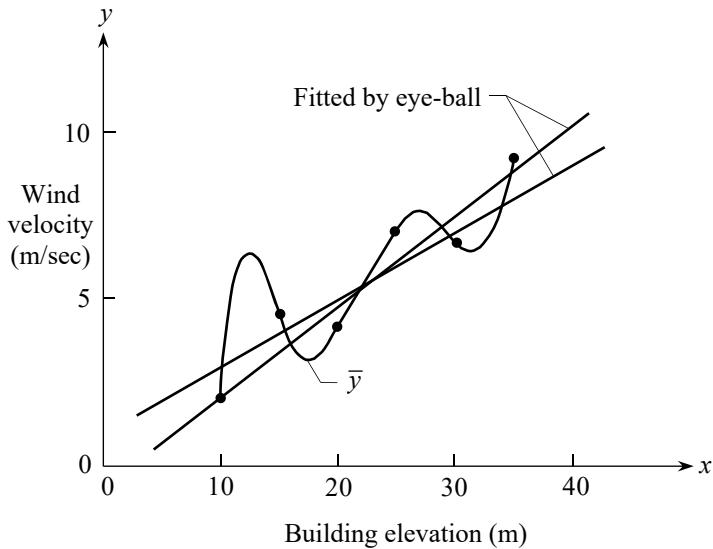


Figure 5.1 Data of the wind velocity that varies with the building elevation fitted by using eye-ball and interpolating function.

Several least-squares regression methods are presented herein. These methods are: (1) the linear regression, (2) the linear regression for nonlinear data, (3) the polynomial regression, and (4) the multiple regression. These methods are explained in details with illustrated examples and computer programs.

5.2 Linear Regression

Linear regression is a simple method for fitting a set of data that tends to vary linearly. Figure 5.2 shows a set of data with n data points, $x_i, y_i, i = 1, 2, \dots, n$. The fitted function is assumed in the form,

$$g(x) = a_0 + a_1 x \quad (5.1)$$

where a_0 and a_1 are the unknown coefficients to be determined later.

As shown in Fig. 5.2, the data y_i at a typical location x_i differs from the value of the fitted function $g(x)$ as $d(x_i)$. The idea behind the least-squares method is to minimize the squares of the differences between the data values and the function values. The total error the occurs from all n data points is

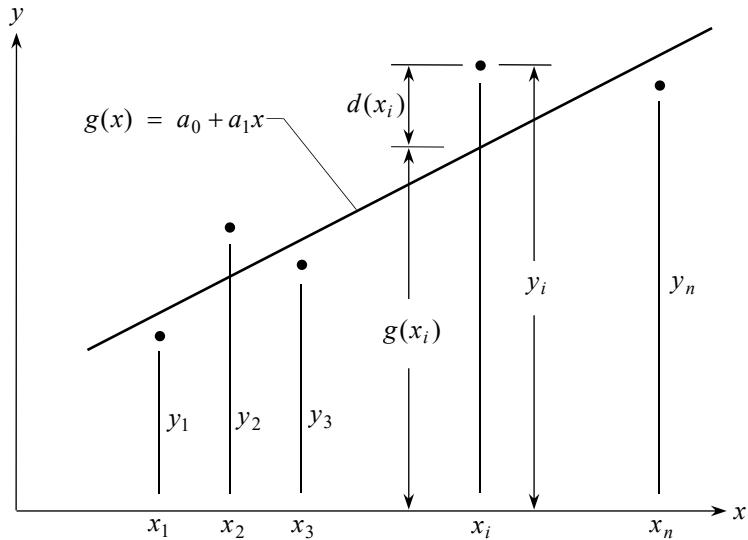


Figure 5.2 Linear regression method for data that tend to vary linearly.

$$E = \sum_{i=1}^n [d(x_i)]^2 \quad (5.2)$$

It is noted that by squaring the differences, the positive differences are not cancelled the negative differences. Equation (5.2) can also be written as

$$E = \sum_{i=1}^n [y_i - g(x_i)]^2 \quad (5.3)$$

By substituting Eq. (5.1) with $x = x_i$, Eq. (5.3) becomes

$$E = \sum_{i=1}^n [y_i - (a_0 + a_1 x_i)]^2 \quad (5.4)$$

The least-squares method is based on results from calculus demonstrating that a function has a minimum value when its partial derivatives are zero. Thus, by performing minimization of the function E in Eq. (5.4) with respect to the unknown coefficients a_0 and a_1 , two equations are obtained as

$$\frac{\partial E}{\partial a_0} = 0 \quad (5.5a)$$

and

$$\frac{\partial E}{\partial a_1} = 0 \quad (5.5b)$$

The partial derivative in Eq. (5.5a) yields

$$2 \sum_{i=1}^n [y_i - (a_0 + a_1 x_i)](-1) = 0$$

$$\begin{aligned} \sum_{i=1}^n y_i - \sum_{i=1}^n a_0 - \sum_{i=1}^n a_1 x_i &= 0 \\ n a_0 + \left(\sum_{i=1}^n x_i \right) a_1 &= \sum_{i=1}^n y_i \end{aligned} \quad (5.6a)$$

Similarly, the partial derivative in Eq. (5.5b) gives

$$\begin{aligned} 2 \sum_{i=1}^n [y_i - (a_0 + a_1 x_i)](-x_i) &= 0 \\ \sum_{i=1}^n x_i y_i - \sum_{i=1}^n a_0 x_i - \sum_{i=1}^n a_1 x_i^2 &= 0 \\ \left(\sum_{i=1}^n x_i \right) a_0 + \left(\sum_{i=1}^n x_i^2 \right) a_1 &= \sum_{i=1}^n x_i y_i \end{aligned} \quad (5.6b)$$

Equations (5.6a) and (5.6b) can be written together in matrix form as

$$\begin{bmatrix} n & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{Bmatrix} \quad (5.7)$$

By using the Cramer's rule explained in section 3.2, the two unknown coefficients a_0 and a_1 can be determined as

$$a_0 = \frac{\left(\sum_{i=1}^n y_i \right) \left(\sum_{i=1}^n x_i^2 \right) - \left(\sum_{i=1}^n x_i y_i \right) \left(\sum_{i=1}^n x_i \right)}{n \left(\sum_{i=1}^n x_i^2 \right) - \left(\sum_{i=1}^n x_i \right)^2} \quad (5.8a)$$

$$a_1 = \frac{n \left(\sum_{i=1}^n x_i y_i \right) - \left(\sum_{i=1}^n x_i \right) \left(\sum_{i=1}^n y_i \right)}{n \left(\sum_{i=1}^n x_i^2 \right) - \left(\sum_{i=1}^n x_i \right)^2} \quad (5.8b)$$

The fitted function $g(x)$ is then obtained by substituting the computed coefficients a_0 and a_1 in Eqs. (5.8a-b) back into Eq. (5.1).

Example 5.1 Employ the linear regression method to establish the best fitted function for the set of the wind velocity and building elevation data in Table 5.1.

The data in Table 5.1 are used to calculate values in Table 5.2 for determining the coefficients a_0 and a_1 of the fitted function according to Eqs. (5.8a-b) as

$$a_0 = \frac{(33.8)(3,475) - (870)(135)}{6(3,475) - (135)^2} = \frac{5}{2,625} = 0.001904$$

$$a_1 = \frac{6(870) - (135)(33.8)}{6(3,475) - (135)^2} = \frac{657}{2,625} = 0.250286$$

Table 5.2 Values required for calculating the two coefficients a_0 and a_1 of the fitted function $g(x)$ in Example 5.1.

x_i	y_i	x_i^2	$x_i y_i$
10	2.2	100	22
15	4.6	225	69
20	4.2	400	84
25	7.0	625	175
30	6.6	900	198
<u>35</u>	<u>9.2</u>	<u>1,225</u>	<u>322</u>
\sum	<u>135</u>	<u>33.8</u>	<u>870</u>

Then, the fitted function $g(x)$ according to Eq. (5.1) is

$$g(x) = 0.001904 + 0.250286x \quad (5.9)$$

Distribution of the fitted function $g(x)$ is plotted to compare with the given data as shown in Fig. 5.3.

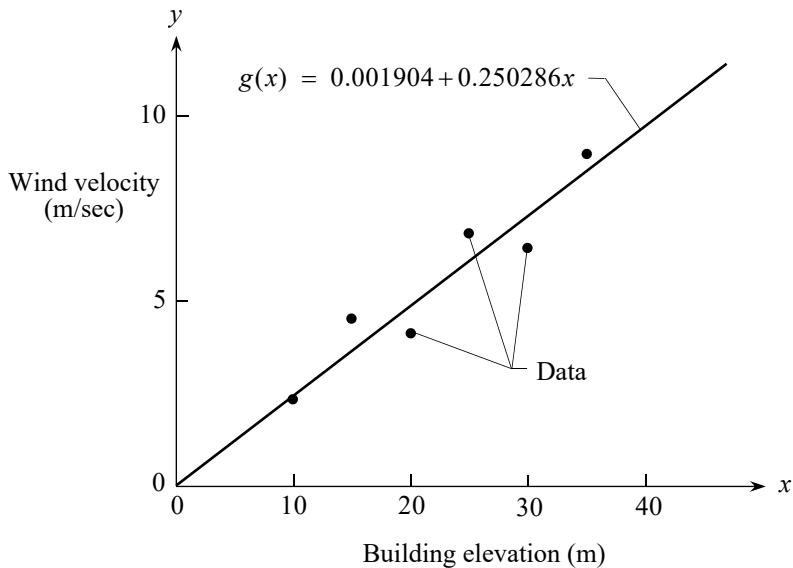


Figure 5.3 Comparison between the fitted function and data of the wind velocity that varies with building elevation.

Example 5.2 Develop a linear regression computer program to determine the two coefficients of the fitted function for n data points. Validate the program by using the data in Example 5.1.

Figure 5.4 shows a linear regression computer program for determining the two coefficients of the fitted function for n data points, $x_i, y_i, i = 1, 2, \dots, n$. Table 5.3 shows the input data required by the program and output of the coefficients a_0 and a_1 for the set of data in Example 5.1.

```
% Program LRegres
% A linear regression program.
% Read number of data set:
fid = fopen('inputL.dat', 'r');
n = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [2 n]);
x = a(1,:); x = x';
y = a(2,:); y = y';
fclose(fid);
% Compute summation terms:
sumx = 0.0; sumy = 0.0;
sumx2 = 0.0; sumxy = 0.0;
for i = 1:n
    sumx = sumx + x(i);
    sumy = sumy + y(i);
    sumx2 = sumx2 + x(i)*x(i);
    sumxy = sumxy + x(i)*y(i);
end
% Fove for the coefficients:
det = n*sumx2 - sumx*sumx;
a0 = (sumy*sumx2 - sumxy*sumx)/det;
a1 = (n*sumxy - sumx*sumy)/det;
fprintf('\nCoefficient a0 = %13.6e', a0)
fprintf('\nCoefficient a1 = %13.6e\n', a1)
```

Figure 5.4 Linear regression computer program for determining the two coefficients a_0 and a_1 of the fitted linear function from n data points.

Table 5.3 Input and output data of the linear regression computer program in Fig. 5.4 for the set of data in Example 5.1.

<u>Input data</u>	<u>Output data</u>
6	Coefficient a0 = .190375E-02
10 2.2	Coefficient a1 = .250286E+00
15 4.6	
20 4.2	
25 7.0	
30 6.6	
35 9.2	

5.3 Linear Regression for Nonlinear Data

Most of data obtained from experiment distribute nonlinearly. These data may be fitted by using the polynomial regression method that will be explained in the next section. The polynomial regression procedure is similar to that of the linear regression except more unknown coefficients are needed to determine. There are sets of data, however, that distribute in some certain patterns. One of the patterns is in the form of the power equation as shown in Fig. 5.5(a). The general form of the power equation is

$$\bar{y} = a \bar{x}^b \quad (5.10)$$

The linear regression method presented in the preceding section can be employed to fit such set of data. The advantage for using the linear regression method to fit the data distribute in the pattern of the power equation is that only two unknown coefficients are needed to determine. The procedure starts from linearization of the power Eq. (5.10) by taking its logarithm to yield

$$\log \bar{y} = \log a + b \log \bar{x} \quad (5.11)$$

The result is in the form

$$y = a_0 + a_1 x \quad (5.12)$$

which is in the same form as Eq. (5.1) as shown in Fig. 5.5(b) where

$$x = \log \bar{x} \quad (5.13a)$$

$$y = \log \bar{y} \quad (5.13b)$$

$$a_0 = \log a \quad (5.13c)$$

$$a_1 = b \quad (5.13d)$$

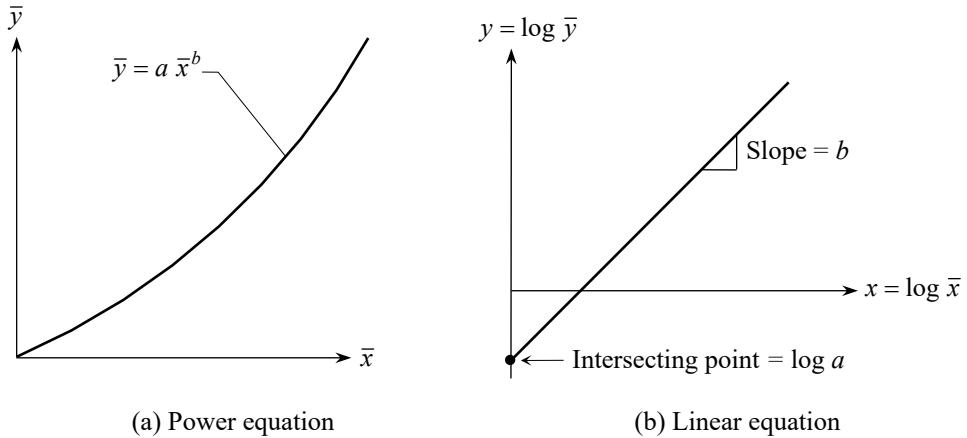


Figure 5.5 Application of the linear regression method to the power equation.

With the fitted function in the linearized form of Eq. (5.12), the linear regression method as explained in section 5.2 can be applied. After the two unknowns coefficients a_0 and a_1 are solved, Eqs. (5.13c-d) are used to obtain the coefficients a and b of the power Eq. (5.10).

\bar{x}	1	2	3	4	5
\bar{y}	0.1	0.7	0.9	1.7	2.1

Example 5.3 Apply the linear regression method to derive the coefficients a and b of the power equation,

$$\bar{y} = a \bar{x}^b$$

by using the set of data given in the table.

Table 5.4 shows the data and their logarithmic values. These values are used to determine the two coefficients a_0 and a_1 of Eq. (5.12) according to Eq. (5.8a-b) as follows,

$$a_0 = \frac{(-0.649)(1.170) - (0.294)(2.079)}{5(1.170) - (2.079)^2} = \frac{-1.371}{1.528} = -0.897$$

$$a_1 = \frac{5(0.294) - (2.079)(-0.649)}{5(1.170) - (2.079)^2} = \frac{2.819}{1.528} = 1.845$$

Table 5.4 The given data and their logarithmic values for applying linear regression method to derive the power equation in Example 5.3.

\bar{x}_i	\bar{y}_i	$x_i = \log \bar{x}_i$	$y_i = \log \bar{y}_i$	x_i^2	$x_i y_i$
1	0.1	0.000	-1.000	0.000	0.000
2	0.7	0.301	-0.155	0.091	-0.047
3	0.9	0.477	-0.046	0.228	-0.022
4	1.7	0.602	0.230	0.362	0.138
5	2.1	0.699	0.322	0.489	0.225
Σ		<u>2.079</u>	<u>-0.649</u>	<u>1.170</u>	<u>0.294</u>

Then, the linearized equation in form of Eq. (5.12) is

$$y = -0.897 + 1.845x \quad (5.14)$$

The computed coefficients a_0 and a_1 are used to determine the coefficients a and b of the power equation according to Eqs. (5.13c-d) as

$$a_0 = \log a \rightarrow a = 0.127 \quad (5.15a)$$

$$a_1 = b \rightarrow b = 1.845 \quad (5.15b)$$

Thus, the fitted power equation is

$$\bar{y} = 0.127 \bar{x}^{1.845} \quad (5.16)$$

Distribution of the fitted power Eq. (5.16) is plotted and compared with the given data points as shown in Fig. 5.6.

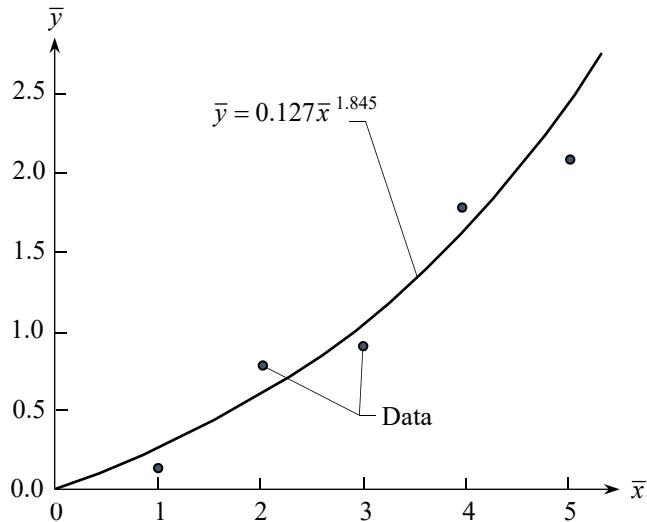


Figure 5.6 Comparison of the fitted power equation using linear regression with the given data in Example 5.3.

There are other types of equations that the linear regression method can be applied to fit a given set of data that distribute in certain patterns. One of the patterns is in form of the exponential model as

$$\bar{y} = a e^{b\bar{x}} \quad (5.17)$$

The exponential model in Eq. (5.17) can also be linearized by taking its natural logarithm to yield

$$\ln \bar{y} = \ln a + b\bar{x} \ln e$$

Since $\ln e = 1$, then

$$\ln \bar{y} = \ln a + b\bar{x} \quad (5.18)$$

Equation (5.18) can be written in the linear form as

$$y = a_0 + a_1 x \quad (5.12)$$

where

$$\begin{aligned} y &= \ln \bar{y} & ; & & x &= \bar{x} \\ a_0 &= \ln a & ; & & a_1 &= b \end{aligned} \quad (5.19)$$

Figure 5.7(a) shows the distribution of the exponential model according to Eq. (5.17) while Fig. 5.17(b) shows the linear distribution of Eq. (5.12). The unknown coefficients a_0 and a_1 of the linearized Eq. (5.12) and the coefficients a and b can be determined in the same way as those of the power equation.

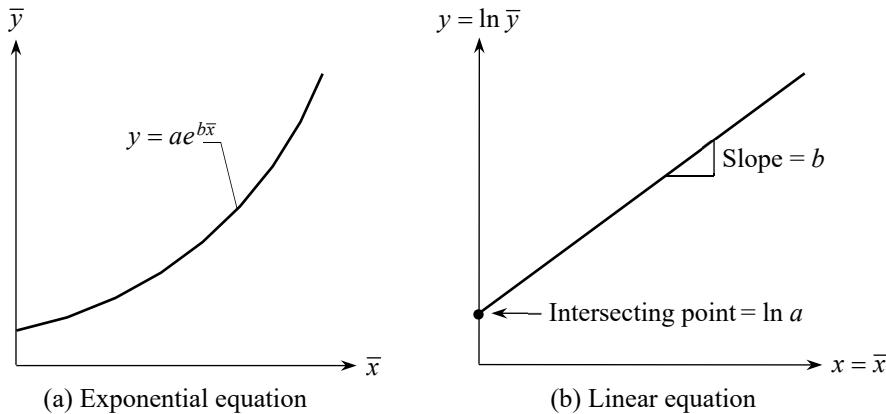


Figure 5.7 Application of the linear regression to the exponential equation.

The saturation-growth-rate equation is another nonlinear equation which is often used to fit a growth data behavior with a limiting condition. The equation is in the form

$$\bar{y} = a \frac{\bar{x}}{b + \bar{x}} \quad (5.20)$$

or

$$\begin{aligned} \frac{b + \bar{x}}{a \bar{x}} &= \frac{1}{\bar{y}} \\ \frac{1}{\bar{y}} &= \frac{1}{a} + \frac{b}{a} \frac{1}{\bar{x}} \end{aligned} \quad (5.21)$$

Equation (5.21) can be written in the linear form as

$$y = a_0 + a_1 x \quad (5.12)$$

where

$$\begin{aligned} y &= 1/\bar{y} & ; & & x &= 1/\bar{x} \\ a_0 &= 1/a & ; & & a_1 &= b/a \end{aligned} \quad (5.22)$$

Figure 5.8(a-b) shows the distributions the saturation-growth-rate equation and the linear equation with its slope and intersection point. Again, the unknown coefficients a_0 and a_1 of the linearized Eq. (5.12) and the coefficients a and b can be determined in the same way as those of the power and exponential equations.

The application of the linear regression method to establish functions for fitting sets of nonlinear data by using the power, exponential and saturation-growth-rate equations is simple. The fitted functions are accurate when the data distributions are in the patterns that could be represented by such equations. If the data distributions are in other patterns, different forms of the fitted functions should be used. The following section explains the polynomial regression method to derived fitted functions for the more general data patterns.

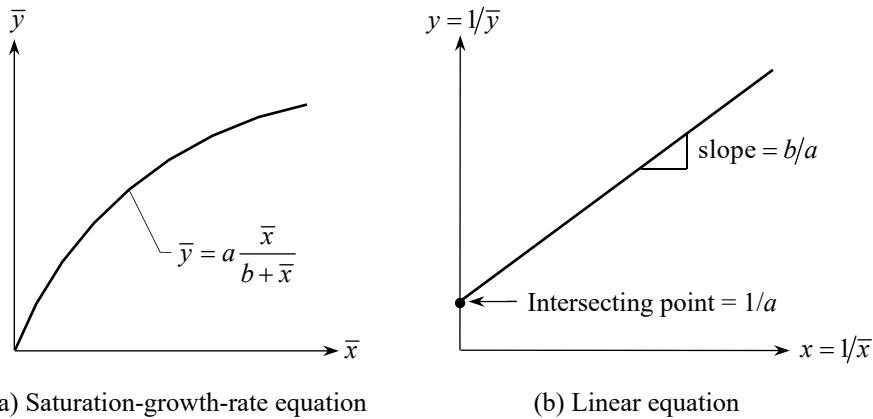


Figure 5.8 Application of the linear regression to the saturation-growth-rate equation.

5.4 Polynomial Regression

The polynomial regression method is suitable to derive a function for fitting a set of data scattered in a polynomial pattern. Figure 5.9 shows a typical fitted polynomial for a set of n data points of $x_i, y_i, i = 1, 2, \dots, n$. A polynomial of order m can be written in the form

$$g(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_m x^m \quad (5.23)$$

where $a_0, a_1, a_2, \dots, a_m$ are the unknown coefficients. These coefficients are determined by first writing the total error E as the sum of the error squares for all data points as

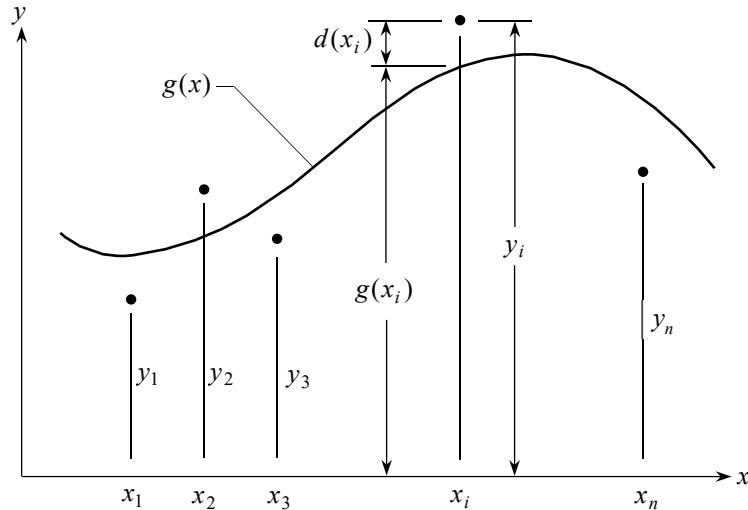


Figure 5.9 Fitting n data points by the polynomial regression method.

$$E = \sum_{i=1}^n [d(x_i)]^2 \quad (5.24)$$

Since the error at each data point is the difference between the data value and the polynomial value, then, the total error E becomes

$$\begin{aligned} E &= \sum_{i=1}^n [y_i - g(x_i)]^2 \\ E &= \sum_{i=1}^n [y_i - (a_0 + a_1 x + a_2 x^2 + \dots + a_m x^m)]^2 \end{aligned} \quad (5.25)$$

The $m+1$ unknown coefficients of $a_0, a_1, a_2, \dots, a_m$ are determined in the same way as those explained in the linear regression method. The total error E is minimized with respect to the unknown coefficients leading to a set of $m+1$ simultaneous equations as

$$\left. \begin{array}{l} \frac{\partial E}{\partial a_0} = 0 \\ \frac{\partial E}{\partial a_1} = 0 \\ \frac{\partial E}{\partial a_2} = 0 \\ \vdots \\ \frac{\partial E}{\partial a_m} = 0 \end{array} \right\} \text{ } m+1 \text{ equations} \quad (5.26)$$

For example, the minimization of the total error E with respect to the unknown a_0 in the first equation of Eq. (5.26) yields

$$\begin{aligned}
 2 \sum_{i=1}^n [y_i - (a_0 + a_1 x_i + a_2 x_i^2 + \dots + a_m x_i^m)](-1) &= 0 \\
 \sum_{i=1}^n y_i - \sum_{i=1}^n a_0 - \sum_{i=1}^n a_1 x_i - \sum_{i=1}^n a_2 x_i^2 - \dots - \sum_{i=1}^n a_m x_i^m &= 0 \\
 n a_0 + \left(\sum_{i=1}^n x_i \right) a_1 + \left(\sum_{i=1}^n x_i^2 \right) a_2 + \dots + \left(\sum_{i=1}^n x_i^m \right) a_m &= \sum_{i=1}^n y_i
 \end{aligned}$$

Similarly, the minimization of the total error E with respect to the unknown a_1 in the second equation of Eq. (5.26) yields

$$\begin{aligned}
 2 \sum_{i=1}^n [y_i - (a_0 + a_1 x_i + a_2 x_i^2 + \dots + a_m x_i^m)](-x_i) &= 0 \\
 \sum_{i=1}^n x_i y_i - \sum_{i=1}^n a_0 x_i - \sum_{i=1}^n a_1 x_i^2 - \sum_{i=1}^n a_2 x_i^3 - \dots - \sum_{i=1}^n a_m x_i^{m+1} &= 0 \\
 \left(\sum_{i=1}^n x_i \right) a_0 + \left(\sum_{i=1}^n x_i^2 \right) a_1 + \left(\sum_{i=1}^n x_i^3 \right) a_2 + \dots + \left(\sum_{i=1}^n x_i^{m+1} \right) a_m &= \sum_{i=1}^n x_i y_i
 \end{aligned}$$

The minimization of the other equations in Eq. (5.26) can be performed in the same fashion leading to a set of $m+1$ simultaneous equations in the matrix form as

$$\left[\begin{array}{cccc|c} n & \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \dots & \sum_{i=1}^n x_i^m \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 & \dots & \sum_{i=1}^n x_i^{m+1} \\ \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^4 & \dots & \sum_{i=1}^n x_i^{m+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_i^m & \sum_{i=1}^n x_i^{m+1} & \sum_{i=1}^n x_i^{m+2} & \dots & \sum_{i=1}^n x_i^{2m} \end{array} \right] \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n x_i^2 y_i \\ \vdots \\ \sum_{i=1}^n x_i^m y_i \end{Bmatrix} \quad (5.27)$$

The square $(m+1) \times (m+1)$ matrix on the left-hand side and the $m+1$ vector on the right-hand side of Eq. (5.27) are known. Thus, the $m+1$ unknown coefficients of $a_0, a_1, a_2, \dots, a_m$ can be solved from the set of simultaneous Eq. (5.27) by using any direct method explained in chapter 3.

Example 5.4 Develop a computer program to derive a polynomial of order m for fitting a set of n data. Then, apply the program to establish a third-order polynomial for fitting the data of the water specific heat that varies with the temperature as shown in Table 5.5. Plot to compare distribution of the fitted polynomial with the given data.

Table 5.5 Data of water specific heat c_p (kJ/kg·°C) that varies with the temperature T (°C).

T	c_p	T	c_p
0	1.00762	55	0.99919
5	1.00392	60	0.99967
10	1.00153	65	1.00024
15	1.00000	70	1.00091
20	0.99907	75	1.00167
25	0.99852	80	1.00253
30	0.99826	85	1.00351
35	0.99818	90	1.00461
40	0.99828	95	1.00586
45	0.99849	100	1.00721
50	0.99878		

Figure 5.10 shows a polynomial regression computer program for determining the unknown coefficients of the fitted m -order polynomial for n data points. The program generates a set of $m+1$ simultaneous equations that is solved by calling the subroutine GAUSS explained in chapter 3. Details of the subroutine GAUSS (not included herein) is shown in Fig. 3.2.

```
% Program PRegres
% A linear regression program.
% Read number of data set:
fid = fopen('inputP.dat', 'r');
n = fscanf(fid, '%f', 1);
d = fscanf(fid, '%f', [2 n]);
x = d(1,:); x = x';
y = d(2,:); y = y';
fclose(fid);
% Read order of polynomial needed
m = input('Enter order of polynomial: ');
b = zeros(m+1,1); a = zeros(m+1,m+1);
% Compute the square matrix on LHS and
% vector on RHS of the system of eqs.:
for ir = 1:m+1
    for ic = 1:m+1
        k = ir + ic - 2;
        for i = 1:n
            a(ir,ic) = a(ir,ic) + x(i)^k;
        end
    end
    for i = 1:n
        b(ir) = b(ir) + y(i)*(x(i)^(ir-1));
    end
end
% Solve the system of equations:
xx = a\b;
% Print out the computed polynomial
% coefficients:
fprintf('\nCoefficients of the fitted')
fprintf(' polynomials are:\n')
for i = 1:m+1
    im1 = i-1;
    fprintf('\n a(%ld) = %13.6e', ...
            im1, xx(i));
end
```

Figure 5.10 Polynomial regression computer program for determining the $m+1$ coefficients of the fitted m -order polynomial from n data points.

The computer program starts from reading an input file that contains the total of $n = 21$ data for this example. The program then establishes a third-order polynomial ($m = 3$) that has $m+1 = 4$ unknown coefficients. These coefficients are solved from a set of $m+1 = 4$ simultaneous equations as shown below

$$\begin{bmatrix} 0.2100000E+02 & 0.1050000E+04 & 0.7175000E+05 & 0.5512500E+07 \\ 0.1050000E+04 & 0.7175000E+05 & 0.5512500E+07 & 0.4516662E+09 \\ 0.7175000E+05 & 0.5512500E+07 & 0.4516662E+09 & 0.3854156E+11 \\ 0.5512500E+07 & 0.4516662E+09 & 0.3854156E+11 & 0.3382122E+13 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{Bmatrix} 0.2102805E+02 \\ 0.1051999E+04 \\ 0.7195141E+05 \\ 0.5531869E+07 \end{Bmatrix}$$

The computed 4 coefficients are

$$\begin{aligned} a_0 &= 0.1006448E+01 & a_1 &= -0.4988565E-03 \\ a_2 &= 0.8460584E-05 & a_3 &= -0.3457691E-07 \end{aligned} \quad (5.28)$$

Thus, the fitted third-order polynomial is

$$\begin{aligned} c_p &= (0.1006448E+01) + (-0.4988565E-03)T \\ &\quad + (0.8460584E-05)T^2 + (-0.3457691E-07)T^3 \end{aligned} \quad (5.29)$$

Distribution of the fitted polynomial in Eq. (5.29) is compared with the given data as shown in Fig. 5.11.

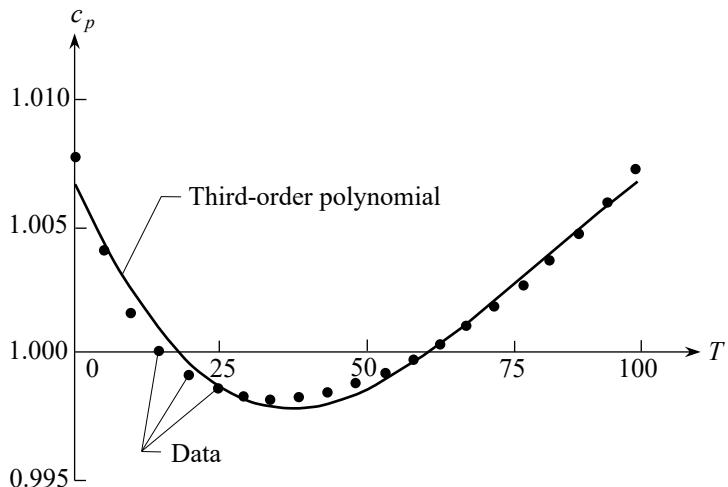


Figure 5.11 Comparison of the fitted third-order polynomial and the given data of the water specific heat that varies with temperature.

5.5 MATLAB Functions for Least-Squares Regression

One of the MATLAB functions used in the least-squares regression method is the `polyfit` function. The command for using the function is

```
p = polyfit(x,y,n)
```

where `x` and `y` are the data
`p` is the vector containing the computed coefficients
`n` is the order of the fitted polynomial

Example 5.5 Employ the `polyfit` function to determine the coefficients of the linear function for fitting the data of the wind velocity and building height as given in Table 5.1.

The data in Table 5.1 are first assigned to store in the variables x and y by using the commands

```
>> x = [10 15 20 25 30 35];
>> y = [2.2 4.6 4.2 7.0 6.6 9.2];
```

The `polyfit` function is then applied by using the first-order polynomial for linear regression as

```
>> a = polyfit(x,y,1)
a =
0.2503    0.0019
```

The computed coefficients are 0.2503 and 0.0019. The first coefficient of 0.2503 is the slope of the fitted function while the second coefficient of 0.0019 is the intersection point on the y -axis. The computed coefficients are identical to those obtained in Example 5.1.

Example 5.6 Employ the `polyfit` function to determine the coefficients of the third-order polynomial for fitting the water specific heat that varies with the temperature by using the set of data in Table 5.5.

Similar to Example 5.5, the data in Table 5.5 are first assigned to store in the variables x and y by using the commands

```
>> x = [0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100];
>> y = [1.00762 1.00392 1.00153 1 0.99907 0.99852 0.99826 0.99818 ...
0.99828 0.99849 0.99878 0.99919 0.99967 1.00024 1.00091 1.00167 ...
1.00253 1.00351 1.00461 1.00586 1.00721];
```

Then, the `polyfit` function with the polynomial order of 3 is used to fit the data,

```
>> format long
>> a = polyfit(x,y,3)
a =
-0.00000003453390    0.00000845381042   -0.00049857114790
1.00644556747600
```

The computed coefficients are identical to those obtained in Example 5.4.

Another valuable function that can be used with the `polyfit` function is `polyval`. The function determines a value between data points from the fitted function. The command for using the `polyval` function is

$$z = \text{polyval}(p, x)$$

where z is the computed value of the function at x location
 p is the vector containing the coefficients of the fitted polynomial
 x is the location for determining the function value

For example, the coefficients of the fitted polynomial in Example 5.6 can be used to determine the polynomial value at $x = 47$ by the commands,

```
>> format short
>> z = polyval(a, 47)

z =
0.9981
```

5.6 Multiple Regression

The preceding sections explain the least-squares regression methods to establish the fitted functions for a set of data. In these methods, the fitted function y depends on a single variable x . For practical problems, the fitted function y may depend on many independent variables x . As an example of a drag force measurement on a car surface in a wind tunnel, the wind pressure y on the car surface varies with the car length x_1 , i.e.,

$$y = y(x_1) \quad (5.30)$$

In addition, the wind pressure varies with the car width x_2 , so that

$$y = y(x_1, x_2) \quad (5.31)$$

Furthermore, the wind pressure on the car surface also depends on the wind speed x_3 ,

$$y = y(x_1, x_2, x_3) \quad (5.32)$$

Thus, in general, the fitted function y are dependent of many variables $(x_1, x_2, x_3, \dots, x_k)$ such that it can be written as

$$y = y(x_1, x_2, x_3, \dots, x_k) \quad (5.33)$$

where k is the number of the independent variables.

5.6.1 Linear

If the data pattern of each independent variable x_j , $j = 1, 2, \dots, k$ (for k independent variables) distributes linearly, the fitted g function can be assumed in the form,

$$g = a_0 + a_1 x_1 + a_2 x_2 + \dots + a_k x_k \quad (5.34)$$

where a_j , $j = 0, 1, 2, \dots, k$ are the unknown coefficients to be determined. The least-squares method is applied by first writing the total error E , which is the summation of the squares of the differences between the fitted function values and the data values, as

$$E = \sum_{i=1}^n [y_i - (a_0 + a_1 x_{1i} + a_2 x_{2i} + \dots + a_k x_{ki})]^2 \quad (5.35)$$

The total error E in Eq. (5.35) is then minimized with respect to the unknown coefficients leading to a set of $k+1$ simultaneous equations as

$$\left. \begin{array}{lcl} \frac{\partial E}{\partial a_0} & = & 0 \\ \frac{\partial E}{\partial a_1} & = & 0 \\ \frac{\partial E}{\partial a_2} & = & 0 \\ \vdots & & \vdots \\ \frac{\partial E}{\partial a_k} & = & 0 \end{array} \right\} \quad k+1 \text{ equations} \quad (5.36)$$

Details of the minimization process is omitted herein and left as an exercise. The minimization process in Eq. (5.36) leads to a set of $k+1$ simultaneous equations written in matrix form as

$$\left[\begin{array}{ccccc} n & \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{2i} & \cdots & \sum_{i=1}^n x_{ki} \\ \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{1i} x_{1i} & \sum_{i=1}^n x_{1i} x_{2i} & \cdots & \sum_{i=1}^n x_{1i} x_{ki} \\ \sum_{i=1}^n x_{2i} & \sum_{i=1}^n x_{1i} x_{2i} & \sum_{i=1}^n x_{2i} x_{2i} & \cdots & \sum_{i=1}^n x_{2i} x_{ki} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_{ki} & \sum_{i=1}^n x_{1i} x_{ki} & \sum_{i=1}^n x_{2i} x_{ki} & \cdots & \sum_{i=1}^n x_{ki} x_{ki} \end{array} \right] \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_k \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_{1i} y_i \\ \sum_{i=1}^n x_{2i} y_i \\ \vdots \\ \sum_{i=1}^n x_{ki} y_i \end{Bmatrix} \quad (5.37)$$

where the square $(k+1) \times (k+1)$ matrix on the left-hand side and the $(k+1) \times 1$ vector on the right-hand side of the Eq. (5.37) are known. Equation (5.37) is then solved to obtain the unknown coefficients $a_0, a_1, a_2, \dots, a_k$ for $k+1$ values by using any direct method described in chapter 3.

Example 5.7 Develop a computer program using the multiple regression method to obtain a fitted function g for a set of data that varies with k independent variables. Test the program on a set of 6 data ($n = 6$) that varies with 2 independent variables ($k = 2$) generated from the equation

$$y = 1 + 2x_1 + 3x_2 \quad (5.38)$$

The 6 data generated from Eq. (5.38) are shown in the table below

i	x_{1i}	x_{2i}	y_i
1	0	0	1
2	0	1	4
3	1	0	3
4	1	2	9
5	2	1	8
6	2	2	11

The generated data from the tested function in the table above lead to the values in the matrices of the simultaneous Eq. (5.37) as

<i>i</i>	x_{1i}	x_{2i}	y_i	$x_{1i}x_{1i}$	$x_{1i}x_{2i}$	$x_{2i}x_{2i}$	$x_{1i}y_i$	$x_{2i}y_i$
1	0	0	1	0	0	0	0	0
2	0	1	4	0	0	1	0	4
3	1	0	3	1	0	0	3	0
4	1	2	9	1	2	4	9	18
5	2	1	8	4	2	1	16	8
6	<u>2</u>	<u>2</u>	<u>11</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>22</u>	<u>22</u>
Σ	<u>6</u>	<u>6</u>	<u>36</u>	<u>10</u>	<u>8</u>	<u>10</u>	<u>50</u>	<u>52</u>

So that the set of simultaneous equations is

$$\begin{bmatrix} 6 & 6 & 6 \\ 6 & 10 & 8 \\ 6 & 8 & 10 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \end{Bmatrix} = \begin{Bmatrix} 36 \\ 50 \\ 52 \end{Bmatrix} \quad (5.39)$$

The set of simultaneous equations is solved to give the unknown coefficients of

$$a_0 = 1, \quad a_1 = 2, \quad a_2 = 3 \quad (5.40)$$

Thus, the fitted function according to Eq. (5.34) is

$$\begin{aligned} g &= a_0 + a_1 x_1 + a_2 x_2 \\ &= 1 + 2x_1 + 3x_2 \end{aligned} \quad (5.41)$$

which is identical to the tested function that was used to generate the data. The distribution of the fitted function is in form of a flat plane as shown in Fig. 5.12. With this fitted function, other values that are not on the given data points can be determined. For example, the value at $x_1 = x_2 = 1$ is

$$g(x_1 = 1, x_2 = 1) = 1 + 2(1) + 3(1) = 6$$

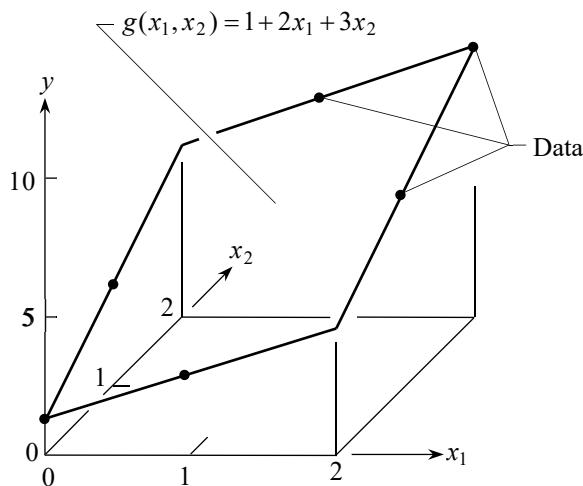


Figure 5.12 Distribution of the fitted function g obtained from the multiple linear regression of the data y that varies with the two independent variables x_1 and x_2 .

Figure 5.13 shows a computer program for the multiple linear regression method. The program starts from reading the set of n input data points. Each data point contains the values $x_{1i}, x_{2i}, \dots, x_{ki}$ of the k independent variables together with the value y_i . With these data, the program establishes a set of simultaneous equations in the form of Eq. (5.37). The set of simultaneous equations is then solved for the unknown coefficients $a_j, j = 0, 1, 2, \dots, k$.

```
% Program MRegres
% A Multiple regression program.
% Read number of data set n,
% number of independent variable k,
% and data of x(i,k) and y(i):
fid = fopen('inputM.dat', 'r');
n = fscanf(fid, '%f', 1);
k = fscanf(fid, '%f', 1);
p = fscanf(fid, '%f', [k+1 n]); p = p';
x(:,1) = squeeze(p(:,1));
x(:,2) = squeeze(p(:,2));
y = squeeze(p(:,3));
fclose(fid);
% Compute the square matrix on LHS and
% vector on RHS of system of eqs.:
a = zeros(k+1,k+1); b = zeros(k+1,1);
for i = 1:n
    for ir = 1:k+1
        if ir == 1
            fr = 1.;
        end
        if ir > 1
            fr = x(i,ir-1);
        end
        for ic = 1:k+1
            if ic == 1
                fc = 1.;
            end
            if ic > 1
                fc = x(i,ic-1);
            end
            a(ir,ic) = a(ir,ic) + fr*fc;
        end
        b(ir) = b(ir) + fr*y(i);
    end
end
xx = a\b;
% Solve the system of equations and
% print out the computed coefficients:
fprintf('\nCoefficients of the fitted')
fprintf(' function are:')
for i = 1:k+1
    iml = i-1;
    fprintf('\n a(%ld) = %13.7e', ...
            iml, xx(i));
end
```

Figure 5.13 Computer program for multiple linear regression method.

Table 5.6 shows the 6 input data of the function y in Eq. (5.38) for Example 5.7. The table also shows the 3 computed coefficients which are identical to those obtained in Eq. (5.40).

Table 5.6 Input and output data of the multiple linear regression computer program in Fig. 5.13 for the set of data in Example 5.7.

<u>Input data</u>	<u>Output data</u>
6 2	Coefficients of the fitted function are:
0 0 1	a(0) = .1000000E+01
0 1 4	a(1) = .2000000E+01
1 0 3	a(2) = .3000000E+01
1 2 9	
2 1 8	
2 2 11	

Understanding the behavior of the data pattern can improve the accuracy of the fitted function. For example, the power functions should be used if the data pattern distributes in the power form. In this case, the fitted function y may be assumed to depend on the 3 independent variables x_1, x_2 and x_3 as

$$\bar{y} = a \bar{x}_1^b \bar{x}_2^c \bar{x}_3^d \quad (5.42)$$

The coefficients a, b, c, d in Eq. (5.42) are unknowns to be determined. The fitted function in Eq. (5.42) is then linearized by taking its logarithm to yield

$$\log \bar{y} = \log a + b \log \bar{x}_1 + c \log \bar{x}_2 + d \log \bar{x}_3 \quad (5.43)$$

which can be written in the form

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 \quad (5.44)$$

Herein, by matching Eqs. (5.44) and (5.43),

$$\begin{aligned} y &= \log \bar{y} ; \quad a_1 = b ; \quad a_2 = c ; \quad a_3 = d \\ a_0 &= \log a ; \quad x_1 = \log \bar{x}_1 ; \quad x_2 = \log \bar{x}_2 ; \quad x_3 = \log \bar{x}_3 \end{aligned} \quad (5.45)$$

Minimization of total error E of the fitted function y in Eq. (5.44) leads to a set of 4 simultaneous equations in the form of Eq. (5.37) with the 4 unknowns of a_0, a_1, a_2 and a_3 . Once these unknowns are determined, the coefficients a, b, c, d can then be obtained from Eq. (5.45). The example below shows the application of the multiple regression method with a fitted power equation for a practical problem.

Example 5.8 Insulating tiles are used as the thermal protection system for the space shuttle. These tiles, which have the size of approximately 20×20 cm and thickness of 6 to 10 cm, are placed beneath the shuttle body and wings. The tiles are subjected to high aerodynamic heating rate during descending at hypersonic speed through the earth atmosphere. Proper gaps must be provided between these tiles to allow their expansions from high temperature. Typical tile layout is shown in Fig. 5.14 with the gap width of w . With the layout of the tiles, there was a concern that a hot gas from the high-speed flow may pass through the gap and creates an excessive heating rate at location A on the adjacent tile as shown in the figure. An experiment was established in a high-speed wind tunnel to measure the heating rate, especially at that location. Table 5.7 shows the measured heating rate data at location A which varies with several flow parameters.

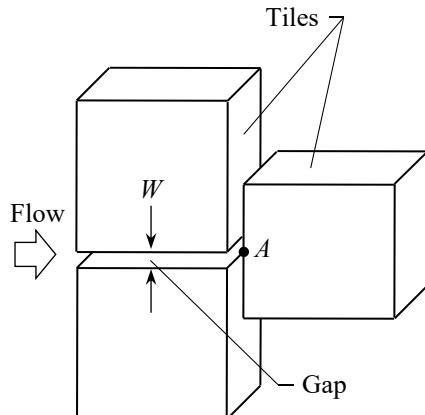


Figure 5.14 Typical layout of space shuttle tiles with gap.

Table 5.7 Measured heating rates at location *A* in Fig. 5.14 from hypersonic flow past shuttle tiles.

<i>q</i>	<i>δ</i>	<i>w</i>	<i>Re</i>	<i>P</i>
2.53	1.13	0.18	14.2×10^5	0.02
2.49	1.01	0.18	4.0×10^5	0.03
2.15	1.01	0.10	8.3×10^5	0.03
1.95	1.01	0.10	4.2×10^5	0.03
3.80	1.01	0.30	8.3×10^5	0.03
2.00	1.01	0.30	4.2×10^5	0.03
3.45	1.01	0.41	8.8×10^5	0.03
2.99	1.01	0.41	4.4×10^5	0.03

Study of the data distribution has shown that the heating rate at location *A* of the tile in Fig. 5.14 varies with the four parameters in the form

$$q = a \left(\frac{\delta}{w} \right)^b Re^c P^d \quad (5.46)$$

where *q* is the heating rate, *δ* is the boundary layer thickness, *Re* is the Reynolds number and *P* is the pressure.

To determine the coefficients *a*, *b*, *c* and *d*, Eq. (5.46) is first linearized by taking its logarithm to yield

$$\log q = \log a + b \log \left(\frac{\delta}{w} \right) + c \log Re + d \log P \quad (5.47)$$

which can be further written in the form

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 \quad (5.48)$$

Then, the given data in Table 5.7 are transformed so that they can be used with the multiple linear regression program in Fig. 5.13 as follows

<i>y = log q</i>	<i>x₁ = log(δ/w)</i>	<i>x₂ = log(Re)</i>	<i>x₃ = log(P)</i>
0.40312	0.79781	6.15229	-1.69897
0.39620	0.74905	5.60206	-1.52288
0.33244	1.00432	5.91908	-1.52288
0.29003	1.00432	5.62325	-1.52288
0.57978	0.52720	5.91908	-1.52288
0.30103	0.52720	5.62325	-1.52288
0.53782	0.39154	5.99445	-1.52288
0.47567	0.39154	5.64345	-1.52288

With the transformed data above, the computer program in Fig. 5.13 yields the values for the coefficients *a₀*, *a₁*, *a₂* and *a₃* as

$$\begin{aligned} a_0 &= -0.400726 & ; & \quad a_2 = 0.326507 \\ a_1 &= -0.272717 & ; & \quad a_3 = 0.581139 \end{aligned} \quad (5.49)$$

Then, by using the relations in Eq. (5.45), the unknown coefficients of the fitted function in Eq. (5.42) are

$$\begin{aligned} a &= 0.397442 & ; & \quad c = 0.326507 \\ b &= -0.272717 & ; & \quad d = 0.581139 \end{aligned} \quad (5.50)$$

Thus, the fitted function for the heating rate is

$$q = (0.397442) \left(\frac{\delta}{w} \right)^{-0.272717} Re^{0.326507} P^{0.581139} \quad (5.51)$$

The fitted function can be used to estimate the heating rate for other conditions different from the given data. For example, the heating rate at $\delta = 1.01$ cm, $w = 0.2$ cm, $Re = 4.0 \times 10^5$ and $P = 0.03$ is

$$\begin{aligned} q &= (0.397442) \left(\frac{1.01}{0.2} \right)^{-0.272717} (4.0 \times 10^5)^{0.326507} (0.03)^{0.581139} \\ &= 2.25 \end{aligned}$$

5.6.2 Polynomial

The multiple linear regression method for which the fitted function y varies linearly with the k independent variables of x_j , $j = 1, 2, \dots, k$ was presented in the preceding section. The method can be extended to the cases when the data y varies nonlinearly in the form of polynomials. For example, if the data tend to vary with the cubic and quadratic distributions along x_1 and x_2 , respectively, as shown in Fig. 5.15, the fitted function may be assumed in the form

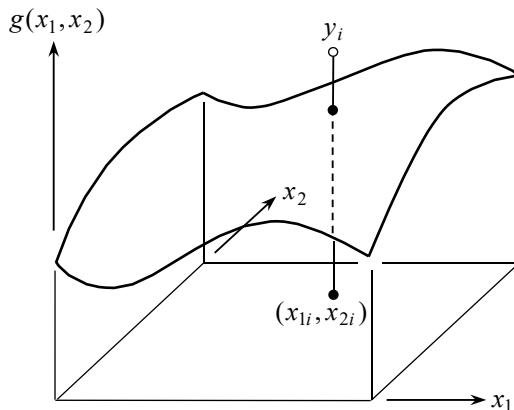


Figure 5.15 A fitted function g that varies with cubic and quadratic distributions along x_1 and x_2 , and a typical data y_i at (x_{1i}, x_{2i}) .

$$g(x_1, x_2) = (b_0 + b_1 x_1 + b_2 x_1^2 + b_3 x_1^3)(c_0 + c_1 x_2 + c_2 x_2^2) \quad (5.52)$$

where b_0, b_1, b_2, b_3 and c_0, c_1, c_2 are the unknown coefficients. The fitted function in Eq. (5.52) can be expanded to yield

$$\begin{aligned} g(x_1, x_2) = & a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + a_4 x_2 + a_5 x_1 x_2 + a_6 x_1^2 x_2 \\ & + a_7 x_1^3 x_2 + a_8 x_2^2 + a_9 x_1 x_2^2 + a_{10} x_1^2 x_2^2 + a_{11} x_1^3 x_2^2 \end{aligned} \quad (5.53)$$

where a_j , $j = 0, 1, \dots, 11$ are the unknown coefficients from the products of b_j and c_j in Eq. (5.52). These unknown coefficients can be determined by the least-squares method as explained in the preceding sections. The total error E is first written as summation of the squares of the difference between the fitted function values and the data values as

$$E = \sum_{i=1}^n [y_i - (a_0 + a_1 x_1 + a_2 x_1^2 + \dots + a_{11} x_1^3 x_2^2)]^2 \quad (5.54)$$

Minimization of the total error E with respect to the 12 unknown constants is then performed leading to a set of 12 simultaneous equations

$$\left. \begin{array}{lcl} \frac{\partial E}{\partial a_0} & = & 0 \\ \frac{\partial E}{\partial a_1} & = & 0 \\ \frac{\partial E}{\partial a_2} & = & 0 \\ \vdots & & \vdots \\ \frac{\partial E}{\partial a_{11}} & = & 0 \end{array} \right\} \text{12 equations} \quad (5.55)$$

Detailed derivation of the simultaneous equations is omitted herein so that it is left for an exercise. The set of 12 simultaneous equations can be written in matrix form as

$$\begin{bmatrix} n & \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{1i}^2 & \dots & \sum_{i=1}^n x_{1i}^3 x_{2i}^2 \\ \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{1i}^2 & \sum_{i=1}^n x_{1i}^3 & \dots & \sum_{i=1}^n x_{1i}^4 x_{2i}^2 \\ \sum_{i=1}^n x_{1i}^2 & \sum_{i=1}^n x_{1i}^3 & \sum_{i=1}^n x_{1i}^4 & \dots & \sum_{i=1}^n x_{1i}^5 x_{2i}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_{1i}^3 x_{2i}^2 & \sum_{i=1}^n x_{1i}^4 x_{2i}^2 & \sum_{i=1}^n x_{1i}^5 x_{2i}^2 & \dots & \sum_{i=1}^n x_{1i}^6 x_{2i}^4 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{11} \end{Bmatrix} = \begin{Bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_{1i} y_i \\ \sum_{i=1}^n x_{1i}^2 y_i \\ \vdots \\ \sum_{i=1}^n x_{1i}^3 x_{2i}^3 y_i \end{Bmatrix} \quad (5.56)$$

where the (12×12) matrix on the left-hand side and the (12×1) vector on the right-hand side of the simultaneous equations above are known. The simultaneous equations are then solved for the 12 unknown coefficients $a_0, a_1, a_2, \dots, a_{11}$. The fitted function $g(x_1, x_2)$ is then obtained by substituting these computed coefficients back into Eq. (5.53).

The multiple regression methods explained in this section suggest that the form for the fitted function should be selected properly. Distribution behavior of the data must be studied prior to selecting an appropriate function for representing them. With the appropriate function, the least-squares method

can be applied and the coefficients of the fitted function are then obtained straight forwardly. It should also be noted that the minimization process presented in this chapter is the fundamental of many advanced numerical methods for analyzing engineering problems.

5.7 Closure

The least-squares method to derive the best fitted function for a given set of data is presented. The method starts from an assumed function that may be in the form of polynomials, power, exponential or some other types. These functions contain the unknown coefficients that are to be determined. The total error between the fitted function and the actual data is then constructed. Such total error is defined as the summation of the squares of the differences between the function values and the data values. The total error is minimized with respect to the unknown coefficients leading to a set of simultaneous equations. The set of simultaneous equation is solved for the unknown coefficients. The computed coefficients are substituted back into the assumed function resulting in the best fitted function.

Several regression methods are presented to fit sets of data that distribute linearly and in the form of higher-order polynomials. The methods are called the linear and polynomial regression methods, respectively. The linear regression method for fitting sets of nonlinear data that distribute in the forms of the power, exponential and saturation-growth-rate equations is also presented. Application of the linear regression method to fit the nonlinear data in the mentioned forms helps reducing complexity of the formulation and the computational effort.

The regression methods above are used to derive the best fitted functions for problems that have only single independent variable. For problems with several independent variables, the multiple regression method is used. The multiple regression method follows the same procedure but leads to more unknown coefficients.

The essential step in the least-squares method is the minimization process. The process seeks the best fitted function by performing partial derivatives of the total error with respect to the problem unknowns and set them to zero. Understanding such process is essential to study other advanced numerical methods for analyzing practical engineering problems.

Exercises

1. Use the least-square regression method to establish a linear function that best fits the data in the table below.

x	0	1	2	4	6	7	9	10	12
y	1	6	10	23	32	39	49	56	65

Compare the computed coefficients of fitted function with those obtained from the computer program in Fig. 5.4. Plot to compare distribution of the fitted function with the data.

2. Use the least-square regression method to establish a linear function that best fits the data in the table below.

x	1,000	1,500	2,000	2,500	3,000	3,500
y	1,457	1,282	932	788	465	264

Compare the computed coefficients of fitted function with those obtained from the computer program in Fig. 5.4. Plot to compare distribution of the fitted function with the data.

3. The data below represent the thermal expansion of a metal that depends on the temperature.

Temperature, °C	40	50	60	70	80	90	100	110
Expansion, %	1.1	1.3	1.3	1.5	1.7	1.9	2.0	2.3

Use the linear regression method to derive the fitted function and estimate values of the thermal expansion at 63°C and 95°C.

4. In the test of a car braking system, the stopping distances are found to depend on the car speeds as shown in the table below.

Velocity, km/hr	10	15	20	30	40	50	60	70	80
Stopping distance, m	5	9	15	18	22	30	35	38	43

Use the linear regression method to derive the best fitted function and estimate the stopping distance when the car speed is 65 km/hr.

5. The expressway toll-fee is determined according to the driving distance. The table below shows the exit numbers, distances and toll-fees.

Exit Number	Distance (Miles)	Toll-fee (Dollars)	Exit Number	Distance (Miles)	Toll-fee (Dollars)
2	10	0.65	8	68	2.45
3	24	1.00	8A	74	2.65
4	35	1.35	9	83	3.05
5	43	1.70	10	87	3.15
6	50	2.05	11	90	3.45
7	53	2.10	12	95	3.80
7A	59	2.25	13	115	4.60

If the expressway authority needs to construct two new exit ways which are 5A and 12A at the distances of 46 and 107 miles, respectively, determine the toll-fees at these new exit ways by using the linear regression method.

6. Distribution of the data given in the table is in the exponential form as

$$\bar{y} = a e^{b\bar{x}}$$

Apply the linear regression method for the nonlinear data to determine the constants a and b of the fitted exponential function above. Then, compare the distribution of fitted function with the data in the table.

\bar{x}	0	1	2	3	4	5	6
\bar{y}	1.2	1.9	2.1	2.8	3.2	4.1	4.9

7. Distribution of the data as shown in the table is in the saturation-growth-rate form as

$$\bar{y} = a \frac{\bar{x}}{b + \bar{x}}$$

Apply the least-squares method to determine the coefficients a and b of the saturation-growth-rate equation above. Repeat the problem by using the second-order polynomial regression. Plot to compare distributions of the two fitted functions with the given data.

\bar{x}	0	1	2	3	4	5	6
\bar{y}	0.02	0.17	0.29	0.34	0.41	0.43	0.47

8. The friction coefficient f of a laminar flow in a tube is found to vary with the Reynolds number Re in the form

$$f = a Re^b$$

Apply the linear regression method to the nonlinear data as shown in the table to determine the unknown constants a and b . Then, use the fitted function to estimate the friction coefficients at the Reynolds numbers of 752 and 1,427

Re	500	1,000	1,500	2,000
f	0.0320	0.0160	0.0107	0.0080

9. The atmospheric pressure P (mm. Hg.) is found to vary with the altitude h (feet) above the sea level in the form

$$P = \alpha e^{-\beta h}$$

Determine the coefficients α and β of the fitted function above from the data in the table below by using the linear regression method. Then, use the fitted function to estimate the pressure at the altitude of 1,250 feet.

P (mm. Hg.)	29.9	29.4	29.0	28.4	27.7
h (feet)	0	500	1,000	1,500	2,000

10. The stress-strain ($\sigma - \varepsilon$) data obtained from testing a concrete column are shown in the table below. The data is best fitted by from the relation

$$\sigma = a\varepsilon e^{-b\varepsilon}$$

Apply the linear regression method for the nonlinear data to determine the coefficients a and b of the fitted equation above. Then, plot to compare distribution of the fitted function with the data.

σ (MPa)	7.1	9.7	11.8	14.4	16.7	19.0	20.7	19.7	18.5
$\varepsilon \times 10^3$	0.265	0.400	0.500	0.700	0.950	1.360	2.080	2.450	2.940

11. Use the polynomial regression method to fit the data in the table below with the second-order polynomial.

x	0	1	3	4	6	8	9	10	11	12
y	1	-7	-17	-19	-17	-7	1	11	23	37

Show the derivation in details. Compare the computed polynomial coefficients with those obtained from the computer program in Fig. 5.10. Then, plot to compare distribution of the fitted polynomial with the data.

12. Fit the data in problem 11 again by using the third-order polynomial. Plot to compare the distributions of the second- and third-order polynomials. Give reasons and comments if the two distributions are the same.

13. From an experiment, the thermal conduction coefficient k of an aluminum material is found to vary with the temperature T as shown in the table. Apply the least-squares method to establish the three fitted functions of the first-, second- and third-order polynomials. Plot to compare distributions of the three fitted polynomials with the experimental data. Then, determine the total error that occurs from each fitted polynomial.

T (°C)	-100	0	100	200	300	400
k (W/m·°C)	215	202	206	215	228	249

14. In Example 5.4, the third-order polynomial is used to fit the water specific heat that varies with the temperature. Employ the computer program in Fig. 5.10 to fit the data in this example again by using the second- and fourth-order polynomials. Plot to compare distributions of the three fitted polynomials. Also, determine the total error that occurs from each fitted polynomial.

15. The air specific heat is found to vary with the temperature in the form of the second-order polynomial according to the data in the table. Use the polynomial regression computer program in Fig. 5.10 to determine the coefficients of the fitted polynomial. Plot to compare distribution of the fitted polynomial with the data. Then, use the fitted polynomial to estimate the values of the specific heat at 1,000 °C 1,500 °C and 2,000 °C.

T (°C)	700	1,200	1,700	2,200	2,700
C_p (kJ/kg·°C)	1.1427	1.2059	1.2522	1.2815	1.2938

16. The table below shows the data of the stress σ that varies with the strain ε . Plot distribution of the data and fit them by an appropriate polynomial. Then, plot to compare distribution of the fitted polynomial with the data. Also, determine the total error of the fitted polynomial from the given data.

σ (MPa)	$\varepsilon \times 10^3$	σ (MPa)	$\varepsilon \times 10^3$
57.7	0.15	383.0	1.66
123.5	0.52	423.0	1.86
191.8	0.76	465.8	2.08
236.0	1.01	497.5	2.27
267.7	1.12	530.6	2.56
309.1	1.42	576.2	2.86
354.0	1.52	613.4	3.19

17. Apply the least-squares regression method to fit data in the table below by using the fourth-order polynomial. Verify the computed polynomial coefficients by comparing with those obtained from the computer program in Fig. 5.10. Then, plot to compare distribution of the fitted polynomial with the given data.

x	1	2	3	4	5	6	7	8	9	10
y	0	2	18	45	100	190	310	505	761	1,127

18. Solve Problem 2 again but by using the MATLAB function `polyfit` with $n = 1$ to establish a linear function. Then, plot to compare the fitted function with the given data.
19. Solve Problem 4 again but by using the MATLAB function `polyfit` with $n = 1$ to establish a linear function. Then, employ the MATLAB function `polyval` to estimate the stopping distance at the car speed of 65 km/hr.
20. Employ the MATLAB `polyfit` function to fit the data in Problem 17 by using the fifth-order polynomial. Then, plot to compare distribution of the fitted polynomial with the given data. Provide comments on the total error of the fitted polynomial as compared to that obtained in Problem 17.
21. From the experimental data as shown in the table below, the pressure head H of a water pump is found to vary with the flow rate Q in the form

$$H = A - BQ^2$$

Apply the linear regression method to fit the nonlinear data and determine the coefficients A and B of the fitted equation above. Then, use the fitted equation to estimate the pressure head at the flow rate of $Q = 260$ m³/h. Solve the problem again but by using the MATLAB functions `polyfit` and `polyval`.

$H(\text{m})$	40.5	37.4	32.9	28.3	23.2	16.8	13.3
$Q(\text{m}^3/\text{h})$	0	115	183	228	274	319	342

22. In the assessment of a water pump performance, the flow rate Q is found to vary with the input power P as shown in the table below.

$P(\text{kW})$	81	78	72	67	64	56	51
$Q(\text{m}^3/\text{h})$	387	349	310	272	231	192	153

Use the MATLAB function `polyfit` to establish an appropriate polynomial for fitting the data above. Then, plot to compare distribution of the fitted polynomial with the given data.

23. The stress-strain (σ - ε) data from the uni-axial tensile testing of a material are shown in the table below.

σ (MPa)	0	229	318	343	360	373
ε (mm/mm)	0	0.0007	0.0012	0.0017	0.0022	0.0027

Plot the distribution of the tested data. Then, employ the MATLAB `polyfit` function with an appropriate polynomial order to fit the data. Plot to compare distribution of the fitted polynomial with the given data. Determine the total error of the fitted polynomial from the data and provide comments on how to reduce such error.

24. Show the derivation of the simultaneous equations in Eq. (5.37) for the multiple linear regression in details. Note that the fitted function y varies linearly with the independent variables $x_i, i = 1, 2, \dots, k$.
25. From the equation of a flat plane, $z = ax + by + c$, determine its values a, b and c to best fit the following data

x	0.4	1.2	3.4	4.1	5.7	7.2	9.3
y	0.7	2.1	4.0	4.9	6.3	8.1	8.9
z	0.03	0.93	3.06	3.35	4.87	5.76	8.92

Then, used the fitted function to estimate the value of z at $x=3.6$ and $y=7.1$.

26. Use the multiple linear regression method as explained in section 5.6 to fit the data in the table below. Show detailed calculation and plot to compare distribution of the fitted function with the given data.

x_1	1	0	2	3	4	2	1
x_2	0	1	4	2	1	3	6
x_3	1	3	1	2	5	3	4
y	4	-5	-6	0	-1	-7	-20

27. Use the multiple linear regression method explained in section 5.6 to fit the data with 4 independent variables as shown in the table below. Verify the computed coefficients of the fitted function with those obtained from the computer program in Fig. 5.13. Plot to compare distribution of the fitted function with the given data.

x_1	3	1	4	0	2	5	1	2
x_2	2	5	1	2	3	4	0	1
x_3	4	2	3	4	1	0	2	3
x_4	0	1	4	3	2	1	2	3
y	-7	7	11	2	11	15	1	5

28. Develop a multiple regression computer program to establish the fitted function y that varies in the forms of the polynomials with order m_1 and m_2 along the independent variables x_1 and x_2 using, respectively. Test the program by using a set of 10 data points ($n = 10$) that are generated from the equation

$$g(x_1, x_2) = (1 + 2x_1 + 3x_1^2)(2 + 3x_2 + 4x_2^2)$$

Verify the computed coefficients obtained from the program with the coefficients in the equation above.

29. From an experiment of heat transfer measurement in a heat exchanger, the Nusselt number Nu is found to vary with the Reynolds number Re and the Prandtl number Pr in the form

$$Nu = \alpha Re^\beta Pr^\gamma r^\delta$$

where r is the ratio of the fluid viscosities at the average and wall temperatures. Apply the multiple regression method to determine the values of α , β , γ and δ for the data in the table below.

Nu	Re	Pr	r
277.0	49,000.0	2.30	0.947
348.0	68,600.0	2.28	0.954
421.0	84,800.0	2.27	0.959
223.0	34,200.0	2.32	0.943
177.0	22,900.0	2.36	0.936
114.8	1,321.0	246.0	0.592
95.9	931.0	247.0	0.583
68.3	518.0	251.0	0.579
49.1	346.0	273.0	0.290
56.0	122.9	1,518.0	0.294
39.9	54.0	1,590.0	0.279
47.0	84.6	1,521.0	0.267
94.2	1,249.0	107.4	0.724
99.9	1,021.0	186.0	0.612
83.1	465.0	414.0	0.512
35.9	54.8	1,302.0	0.273

30. Use the multiple regression method to derive the set of simultaneous equations in Eq. (5.56). The set of simultaneous equations is derived to determine the coefficients of a fitted function that varies in the form of the third- and second-order polynomials along the independent variables x_1 and x_2 , respectively. Develop a corresponding computer program to determine the coefficients of the fitted function by using the data in the table below. The data represent values of the measured temperatures in degree Kelvin at different locations along the outer circumference of a cylinder. The cylinder is placed in a high-speed wind tunnel and is subjected to a hot air flow at the speed of Mach 8. Herein, x_1 represents the angular location in degrees along the cylinder outer circumference and x_2 denotes the time in seconds.

	x_2					
	0	20	40	60	80	100
0	560	700	840	950	1,000	1,080
5	560	680	790	900	950	1,050
10	560	820	920	1,300	1,400	1,500
15	560	710	960	1,180	1,320	1,440
20	560	740	1,080	1,270	1,430	1,530
25	560	760	1,160	1,270	1,390	1,480
x_1	30	560	720	1,140	1,220	1,340
	35	560	700	1,060	1,140	1,220
	40	560	700	980	1,080	1,150
	45	560	670	900	960	1,050
	50	560	660	860	920	990
	55	560	650	810	870	920
	60	560	630	760	820	860
						910

Chapter

6

Numerical Integration and Differentiation

6.1 Introduction

Integration and differentiation always arise in the process of solving scientific and engineering problems. Most of the functions that occur during solving practical problems are complicated and can not be integrated analytically. Numerical integration is thus required to provide approximate solutions. Several numerical integration methods are firstly presented in this chapter. Numerical differentiation methods are explained later at the end of the chapter. Numerical integration and differentiation methods presented in this chapter are the basis to study higher-level numerical methods in the later chapters for solving practical problems.

Performing integration by using integrating formulas has been taught in high-school and early years of undergraduate level. Simple functions can be integrated easily by using standard integrating formulas. For example, the integral of the polynomial function below is,

$$I = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx = \left[\frac{2x^4}{4} - \frac{5x^3}{3} + \frac{3x^2}{2} + x \right]_0^2 = 2 \frac{2}{3} \quad (6.1)$$

Or, the integral of a logarithmic function is,

$$I = \int_1^2 \ln x dx = \left[x \ln x - x \right]_1^2 = 0.386294 \quad (6.2)$$

Integration of some functions yields complicated result, such as

$$\int_0^b \frac{dx}{1+x^4} = \frac{1}{4\sqrt{2}} \ln \left(\frac{b^2 + b\sqrt{2} + 1}{b^2 - b\sqrt{2} + 1} \right) - \frac{1}{2\sqrt{2}} \tan^{-1} \left(\frac{b\sqrt{2}}{b^2 - 1} \right) \quad (6.3)$$

Results of the integrations above are exact and can be used directly.

There are numerous functions that are complicated and occur during solving practical problems. These functions can not be integrated analytically to obtain exact solutions. For example, integration of an error function below is needed in the process for solving the transient temperature response from conduction heat transfer in a bar,

$$I = \int_a^b e^{-x^2} dx \quad (6.4)$$

where a and b are constants. A numerical method is needed to provide solution for the integration of the error function above. In some other problems, such as the behavior of a swinging pendulum, integration of the function shown in Eq. (6.5) is required.

$$K(\theta) = \int_0^{\pi/2} \frac{dx}{\sqrt{1 - \sin^2 \frac{\theta}{2} \sin^2 x}} \quad (6.5)$$

Equation (6.5) is called the elliptic integral of the first kind where θ is the swinging angle. A numerical method is again needed to provide the integral solution. The solutions at different angles are normally tabulated and presented in textbooks so that they can be used conveniently.

Numerical methods presented in this chapter can be applied to integrate a given function easily and conveniently. The function or *integrand* may be in any form, simple or complex function, as shown in the examples above. The general form of the integration is

$$I = \int_a^b f(x) dx \quad (6.6)$$

The integral in Eq. (6.6) means that the function f at the x location is multiplied by dx and summed over the interval $x = a$ to $x = b$. It should be noted that the integral sign \int has the style of the letter S representing the meaning of *Summation*.

From the explanation above, the integration is similar to the multiplication between the height of the function f at the x location and the length dx which leads to a long skinny area as shown in Fig. 6.1(a). These areas are then summed together to yield the total area under the curve of the given function. By developing a computer program and using a very small value of dx , the area under the curve can be determined accurately and effectively. Mathematically, the integral obtained from such process approaches the exact solution as $dx \rightarrow 0$ as shown in Fig. 6.1(b). Determination for the area under the curve, by using several numerical integration methods, is presented in this chapter. Some of these methods provide high solution accuracy while the others are quite simple to use.

Numerical integration methods presented herein are: (1) the trapezoidal rule with single and multiple segments, (2) the Simpson's rule with single and multiple segments, (3) the Romberg integration, and (4) the Gauss quadrature. Multiple integration and numerical differentiation are then explained. The last two topics are important for solving practical problems and are the basis for studying other higher-level numerical methods in the later chapters.

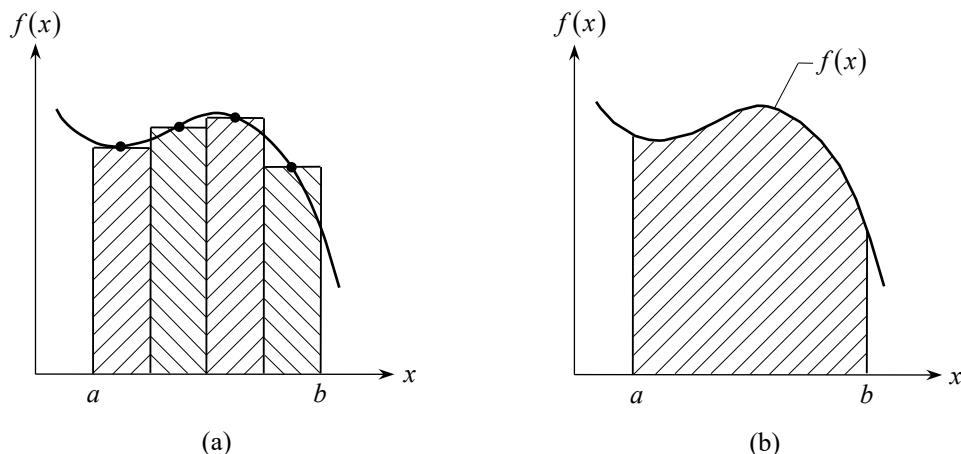


Figure 6.1 Integration of a function represented by area under curve.

6.2 Trapezoidal Rule

The procedure to determine an integral by using the trapezoidal rule is simple and easy to understand. The integral is approximated by the trapezoidal area as shown by Fig. 6.2.

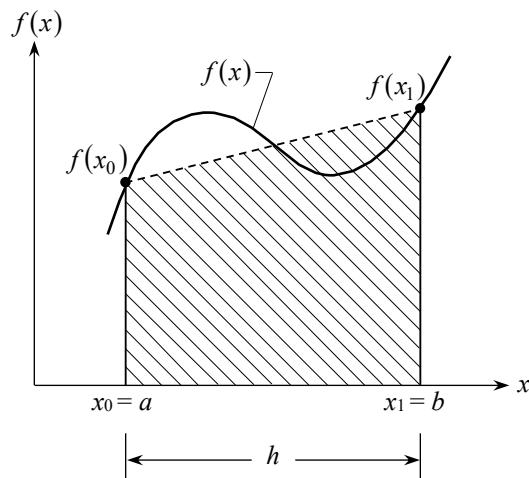


Figure 6.2 Approximate integral by the trapezoidal rule.

Figure 6.2 shows distribution of a function $f(x)$ in the interval $a \leq x \leq b$. The integral of the function within such interval is

$$I = \int_a^b f(x)dx \quad (6.7)$$

The integral solution is represented by the area under the function $f(x)$. The area may be approximated by the trapezoidal area under the dashed line as,

$$\begin{aligned} I &\approx (x_1 - x_0) \frac{f(x_0) + f(x_1)}{2} \\ &= \frac{h}{2} [f(x_0) + f(x_1)] \end{aligned} \quad (6.8)$$

From the figure, $x_1 - x_0 = b - a = h$, then

$$I = \frac{b-a}{2} [f(x_0) + f(x_1)] \quad (6.9)$$

It is noted that the dashed line is the first-order Lagrange polynomial as shown in Eq. (4.20). If Eq. (4.20) is substituted into Eq. (6.7),

$$I \approx \int_a^b \left[\frac{x_1 - x}{x_1 - x_0} f(x_0) + \frac{x_0 - x}{x_0 - x_1} f(x_1) \right] dx \quad (6.10)$$

and the integration is performed, then

$$I = \frac{h}{2} [f(x_0) + f(x_1)] \quad (6.11)$$

The integral is identical to that obtained in Eq. (6.8). The above procedure also suggests that high order Lagrange polynomials can be used in order to provide more accurate integral solutions.

Example 6.1 Use the trapezoidal rule to estimate the integral

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \quad (6.12)$$

Compare the computed solution with the exact solution. Also determine the true error and true percentage error.

Distribution of the given function $f(x)$ in Eq. (6.12) is shown in Fig. 6.3.

Herein,

$$\begin{aligned} x_0 &= a = 0 & ; & f(x_0) = 0 - 0 + 0 + 1 = 1 \\ x_1 &= b = 2 & ; & f(x_1) = 16 - 20 + 6 + 1 = 3 \end{aligned}$$

Then, the approximate integral from Eq. (6.8) or (6.9) which is represented by the area under the dashed line is

$$\begin{aligned} I &= \frac{h}{2} [f(x_0) + f(x_1)] \\ &= \frac{2-0}{2} (1+3) \\ I &= 4 \end{aligned} \quad (6.13)$$

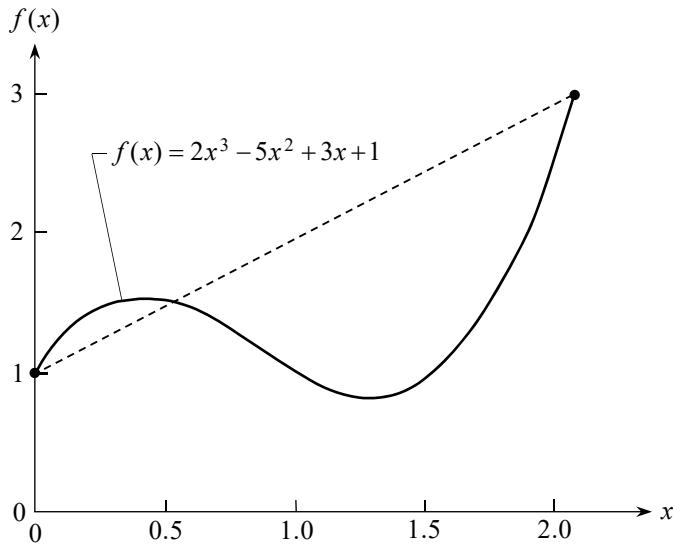


Figure 6.3 Use of trapezoidal rule to obtain an approximate integral which is the area under the dashed line.

It is noted that the exact integral is

$$\begin{aligned} I &= \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx = \left[\frac{2x^4}{4} - \frac{5x^3}{3} + \frac{3x^2}{2} + x \right]_0^2 \\ &= \frac{8}{3} = 2.666667 \end{aligned} \quad (6.14)$$

Thus, the true error is

$$E_t = \frac{8}{3} - 4 = -1.333333 \quad (6.15)$$

and the true percentage error is

$$\varepsilon_t = \frac{\frac{8}{3} - 4}{\frac{8}{3}} \times 100\% = -50\% \quad (6.16)$$

Since the exact integral is not available in general, the accuracy of the computed integral obtained from the trapezoidal rule is not known. The error that arises by using the trapezoidal rule, which will be shown later, is

$$E_t = -\frac{1}{12} f''(\xi)(b-a)^3 \quad (6.17)$$

where ξ is a value between the integration limits a and b . Equation (6.17) implies that the trapezoidal rule provides exact integral if the given function $f(x)$ is linear (f' is constant and f'' vanishes). Understanding the error statement in Eq. (6.17) can lead to an improved integral solution. Such understanding is a basis of the Romberg integration method that will be explained in Section 6.7 for providing more accurate integrals.

The integration error that arises by using the trapezoidal rule as shown in Eq. (6.17) can be derived from the Taylor's series. The Taylor's series as shown in Eq. (2.29) is firstly written as,

$$\begin{aligned} f(x) &= f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \dots \\ &\quad + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n \end{aligned} \quad (6.18)$$

where R_n denotes the remainder that consists the remaining terms of the infinite series. The remainder can be written in the form

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \quad (6.19)$$

where ξ is an unknown value between x_0 to x . For example, the function $f(x)$ can be determined if the function and its first derivative at point a are known. So that Eq. (6.18) becomes

$$f(x) = f(a) + f'(a)(x - a) + R_1 \quad (6.20)$$

where R_1 is the remainder that consists of the second to the infinite terms. In this case, the remainder from Eq. (6.19) is

$$R_1 = \frac{f''(\xi)}{2!}(x - x_0)^2 \quad (6.21)$$

Equations (6.20) and (6.21) imply that the exact value of $f(x)$ can be obtained if a value of ξ is selected properly.

The polynomial function of order n passing through $n+1$ data points was derived in Eq. (4.10) of Chapter 4. To fit a function $f(x)$ by a polynomial function, the remainder similar to Eq. (6.18) must be included as follow

$$\begin{aligned} f(x) &= C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + \dots \\ &\quad + C_n(x - x_0)(x - x_1)\dots(x - x_{n-1}) + R_n \end{aligned} \quad (6.22)$$

where the coefficients $C_i, i = 0, 1, 2, \dots, n$ are determined by applying the conditions at the locations $x_0, x_1, x_2, \dots, x_n$. These coefficients are identical to those shown in Eq. (4.11) as follows,

$$C_0 = f(x_0) \quad (6.23a)$$

$$C_1 = \frac{f(x_1) - f(x_0)}{h} = \frac{\Delta f(x_0)}{h} \quad (6.23b)$$

$$C_2 = \frac{f(x_2) - 2f(x_1) + f(x_0)}{2h^2} = \frac{\Delta^2 f(x_0)}{2! h^2} \quad (6.23c)$$

through C_n , where the symbol Δ refers to the forward differencing. In this case, the remainder in Eq. (6.22) is

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1)\dots(x - x_{n-1})(x - x_n) \quad (6.24)$$

which is in the same form as that of the Taylor's series in Eq. (6.19). Inclusion of the remainder to the expression of Eq. (6.22) leads to the exact value of function $f(x)$ if the value of ξ between x_0 and x_n is selected properly.

As an example of a simple case shown in Fig. 6.2, Eq. (6.22) with $x_0 = a$ and $x_1 = b$ is

$$\begin{aligned} f(x) &= C_0 + C_1(x - x_0) + R_1 \\ &= f(a) + \frac{\Delta f(a)}{h}(x - a) + \frac{f''(\xi)}{2!}(x - a)(x - b) \end{aligned} \quad (6.25)$$

By substituting Eq. (6.25) which is the exact expression for any function $f(x)$ into the integral Eq. (6.7) and performing integration,

$$\begin{aligned} I &= \int_a^b \left[f(a) + \frac{\Delta f(a)}{h}(x - a) + \frac{f''(\xi)}{2!}(x - a)(x - b) \right] dx \\ &= \left[f(a)x + \frac{\Delta f(a)}{h} \left(\frac{x^2}{2} - ax \right) + \frac{f''(\xi)}{2!} \left(\frac{x^3}{3} - \frac{ax^2}{2} - \frac{bx^2}{2} + abx \right) \right]_a^b \\ I &= f(a)(b - a) + \frac{\Delta f(a)}{h} \frac{(b - a)^2}{2} + \frac{f''(\xi)}{2!} \left(-\frac{(b - a)^3}{6} \right) \end{aligned}$$

Since $(b - a) = h$ and $\Delta f(a) = f(b) - f(a)$ from Eq. (6.23b), then

$$\begin{aligned} I &= f(a)h + \frac{f(b) - f(a)}{h} \frac{h^2}{2} - \frac{1}{12} f''(\xi)h^3 \\ I &= \frac{h}{2} (f(b) + f(a)) - \frac{1}{12} f''(\xi)h^3 \end{aligned} \quad (6.26)$$

The first term in Eq. (6.26) is the approximate integral by using the trapezoidal rule while the second term represents the error. Thus, the error that occurs from the trapezoidal rule is

$$E_t = -\frac{1}{12} f''(\xi)h^3 = -\frac{1}{12} f''(\xi)(b - a)^3 \quad (6.27)$$

Example 6.2 The exact integral of the function

$$f(x) = 2x^3 - 5x^2 + 3x + 1 \quad (6.28)$$

from the limit $a = 0$ to $b = 2$ as shown in Example 6.1 is 2.666667. The approximate solution by using the trapezoidal rule is 4 and the exact error is -1.333333. If the exact integral is not available, the error may be determined as follows.

From Eq. (6.27), the error from using the trapezoidal rule is

$$E_t = -\frac{1}{12} f''(\xi)h^3 \quad (6.27)$$

Herein, $h = b - a = 2$ and the derivatives of the function $f(x)$ are

$$f'(x) = 6x^2 - 10x + 3 \quad (6.29a)$$

$$f''(x) = 12x - 10 \quad (6.29b)$$

If the location ξ is chosen at $x = 1$ which is at the middle of the interval between $a = 0$ to $b = 2$, then

$$f''(\xi) = 12(1) - 10 = 2$$

Thus, the error from Eq. (6.27) is

$$E_a = -\frac{1}{12}(2)(2)^3 = -1.333333 \quad (6.30)$$

The approximate error as shown in Eq. (6.30) is exact. The example demonstrates that the exact error can be obtained if the location ξ is chosen properly. Because the location ξ for the exact error is not known, the average value for second derivative of the function is determined instead as

$$\begin{aligned} f''(\xi) &= \int_0^2 (12x - 10) dx = \left[\frac{12x^2}{2} - 10x \right]_0^2 \\ &= 24 - 20 = 4 \end{aligned}$$

Thus, the approximate error is

$$E_a = -\frac{1}{12}(4)(2)^3 = -2.666667 \quad (6.31)$$

which has the same sign and order of magnitude as the exact error. It is noted that understanding of the approximate error above is useful to evaluate the accuracy of the computed solution by other methods in the later sections.

6.3 Composite Trapezoidal Rule

The trapezoidal method presented in the preceding section can provide an improved integral solution if the integration interval is divided into many segments so that the trapezoidal rule is applied to each segment. The computed areas from these segments are then combined to yield the integral solution for the entire interval. The procedure is called *composite* or *multiple-application trapezoidal rule* as shown by Fig. 6.4.

Figure 6.4 shows distribution of a typical function $f(x)$ between the interval $a \leq x \leq b$. The interval from a to b is divided into n segments with equal width of h ,

$$h = \frac{b-a}{n} \quad (6.32)$$

The coordinates at both ends of the segments are

$$x_i = x_0 + i h \quad i = 0, 1, 2, \dots, n \quad (6.33)$$

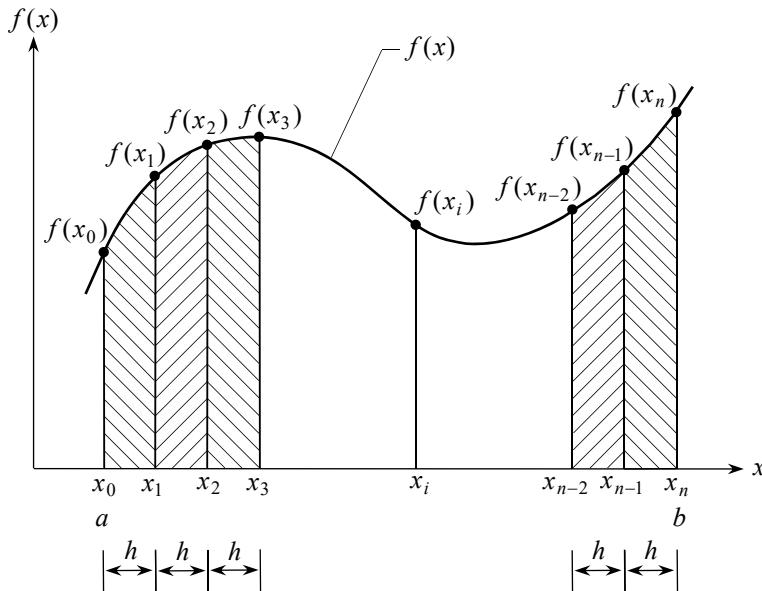


Figure 6.4 Approximate integral determination by composite trapezoidal rule.

Integration of the function $f(x)$ between $a \leq x \leq b$ is performed by first dividing the entire interval into n segments starting from the segments with $x_0 \leq x \leq x_1$, $x_1 \leq x \leq x_2$ until $x_{n-1} \leq x \leq x_n$ as,

$$\begin{aligned} I &= \int_a^b f(x) dx \\ I &= \int_{x_0}^{x_1} f(x) dx + \int_{x_1}^{x_2} f(x) dx + \dots + \int_{x_{n-1}}^{x_n} f(x) dx \end{aligned} \quad (6.34)$$

Then, the trapezoidal rule is applied to each segment that has the width of h as follows,

$$\begin{aligned} I &\approx \frac{h}{2} (f(x_0) + f(x_1)) + \frac{h}{2} (f(x_1) + f(x_2)) + \dots + \frac{h}{2} (f(x_{n-1}) + f(x_n)) \\ &= \frac{h}{2} (f(x_0) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1}) + f(x_n)) \\ I &= \frac{h}{2} \left(f(x_0) + f(x_n) + 2 \sum_{i=1}^{n-1} f(x_i) \right) \end{aligned} \quad (6.35)$$

Equation (6.35) can be used to develop a corresponding computer program directly. The program can be employed to find approximate integral of a given function conveniently. A more accurate solution is obtained by simply increasing the number of segments n in the program.

The error arisen by using the composite trapezoidal rule that divides the entire interval from the limit a to b into n segments is

$$E_t = -\frac{1}{12} \left(\frac{b-a}{n} \right)^3 \sum_{i=1}^n f''(\xi_i) \quad (6.36)$$

where $f''(\xi_i)$ is the second derivative of the function at location ξ_i of the segment i . The second derivative value of the function is different from segment to segment. Their average value is

$$\bar{f}'' \approx \frac{1}{n} \left(\sum_{i=1}^n f''(\xi_i) \right) \quad (6.37)$$

Thus, the total approximate error according to Eq. (6.36) becomes

$$E_a = -\frac{1}{12} \frac{(b-a)^3}{n^2} \bar{f}'' \quad (6.38)$$

The term n^2 on the numerator of Eq. (6.38) implies that the total error will reduce four times if number of the segments is double. Understanding the behavior of the error reduction above can help verifying the accuracy of the computed integral solutions.

Example 6.3 Apply the composite trapezoidal rule to estimate the integral

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \quad (6.12)$$

which is in the same form as Example 6.1 but by developing a computer program. The program can vary the number of segments so that accuracy of the computed solutions can be studied by comparing them with the exact solution.

To clearly demonstrate the use of the composite trapezoidal rule, the entire interval from $a = 0$ to $b = 2$ is divided into 4 segments ($n = 4$) as shown in Fig. 6.5. The width h of each segment according to Eq. (6.32) is

$$h = \frac{b-a}{n} = \frac{2-0}{4} = 0.5 \quad (6.39)$$

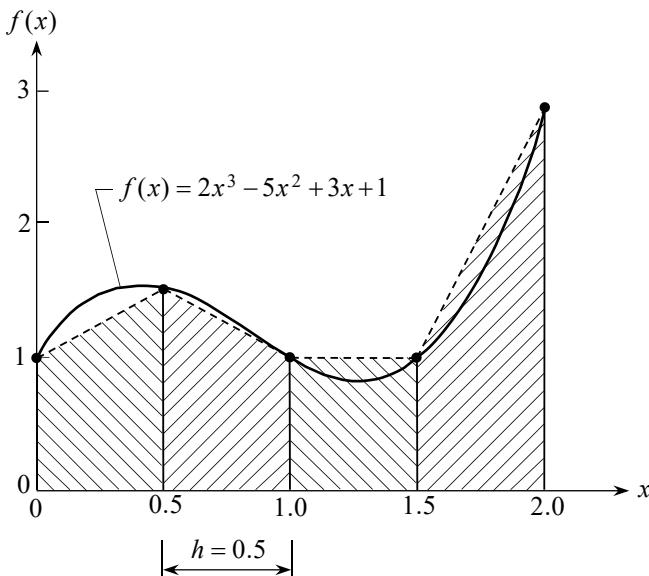


Figure 6.5 Use of the composite trapezoidal rule to approximate the integral in Example 6.3 by dividing the entire interval into 4 segments.

The values of the function $f(x)$ at the ends of the four segments are

$$\begin{aligned}f(x_0 = 0) &= 0 - 0 + 0 + 1 &= 1 \\f(x_1 = 0.5) &= 0.25 - 1.25 + 1.5 + 1 &= 1.5 \\f(x_2 = 1.0) &= 2 - 5 + 3 + 1 &= 1 \\f(x_3 = 1.5) &= 6.75 - 11.25 + 4.5 + 1 &= 1 \\f(x_4 = 2.0) &= 16 - 20 + 6 + 1 &= 3\end{aligned}$$

Thus, the approximate integral, according to Eq. (6.35), by using four segments ($n = 4$) is,

$$\begin{aligned}I &= \frac{h}{2} \left(f(x_0) + f(x_4) + 2 \sum_{i=1}^{4-1} f(x_i) \right) = \frac{0.5}{2} (1 + 3 + 2(1.5+1+1)) \\I &= 2.75\end{aligned}\quad (6.40)$$

Table 6.1 shows the approximate integrals obtained from the computer program in Fig. 6.6 by using the composite trapezoidal rule. The table indicates that the error is reduced by four times as the number of segments is double. The behavior of the error reduction can be observed from the solutions when $n = 2$ and $n = 4$ or when $n = 5$ and $n = 10$.

Table 6.1 Approximate solutions of the integral in Example 6.3 by using the composite trapezoidal computer program in Fig. 6.6 as compared to the exact solution of $I = 2.666667$.

<i>n</i>	<i>h</i>	<i>I</i>	ϵ_t (%)
2	1.0000	3.000000	-12.5
3	0.6667	2.814815	-5.6
4	0.5000	2.750000	-3.1
5	0.4000	2.720000	-2.0
6	0.3333	2.703704	-1.4
7	0.2857	2.693878	-1.0
8	0.2500	2.687500	-0.8
9	0.2222	2.683128	-0.6
10	0.2000	2.680000	-0.5

Figure 6.6 shows a computer program to determine approximate integral of a given function $f(x)$ from a to b by using the composite trapezoidal rule. The number of segments is input by the user. The program can be modified to integrate other functions by simply changing the function statement and the integration limits declared in the program.

```
% Program Trapez
% A multiple-segment trapezoidal program
% for estimating integral of f(x).
func = @(x) (2.*x^3 - 5.*x^2 + 3.*x + 1.);
a = 0.; b = 2.;
% Read the number of segments required:
n = input( ...
    '\nEnter the number of segments: ');
h = (b - a)/n; sum = 0.; x = a + h;
for i = 1:n-1
    fx = func(x); sum = sum + fx; x = x + h;
end
fx0 = func(a); fxn = func(b);
sol = (fx0 + fxn + 2.*sum)*h/2.;
fprintf( ...
    'The computed integral is %10.6f', sol)
```

Figure 6.6 A computer program to determine the integral in Example 6.3 by using the composite trapezoidal rule.

6.4 Simpson's Rule

From the trapezoidal rule explained in section 6.2, the integral value is estimated by using the area under a straight line (dashed line connecting points a and b in Fig. 6.2). Because the distribution of a function to be integrated is arbitrary, the area under the straight line is, in general, not accurate for representing the integral value. In this section, the Simpson's rule is presented for which the integral value is determined from the area under the second-order polynomial (dashed line in Fig. 6.7).

Figure 6.7 shows the distribution of a function $f(x)$ between $a \leq x \leq b$. The objective is to

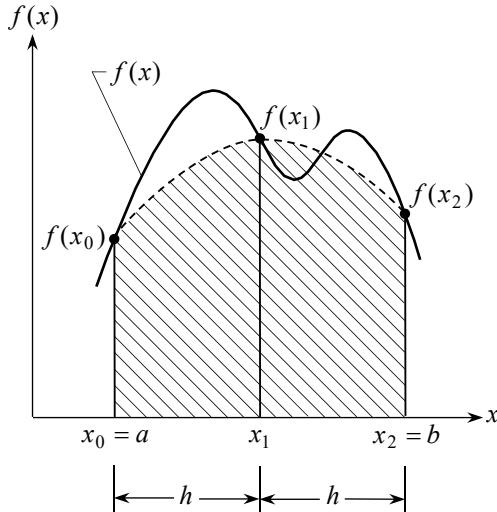


Figure 6.7 Use of the Simpson's 1/3 rule to obtain approximate integral which is the area under the dashed line.

determine the integral

$$I = \int_a^b f(x) dx \quad (6.41)$$

The integral value is determined by approximating the function $f(x)$ in the form of a second-order polynomial. By substituting the second-order Lagrange polynomial as shown in Eq. (4.29) into Eq. (6.41)

$$\begin{aligned} I \approx & \int_a^b \left[\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) \right. \\ & \left. + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2) \right] dx \end{aligned} \quad (6.42)$$

The values of the function $f(x_0), f(x_1), f(x_2)$ can be determined at the locations x_0, x_1, x_2 . If $x_2 - x_1 = x_1 - x_0 = h$, the approximate integral from Eq. (6.42) is

$$I \approx \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] \quad (6.43)$$

where $h = \frac{b-a}{2}$ (6.44)

Equation (6.43) is called the Simpson's one-third rule. It is noted that the term "one-third" refers to the presence of factor 1/3 in front of the expression. Equation (6.43) can also be written in a more general form as

$$I \approx \frac{(b-a)}{6} [f(x_0) + 4f(x_1) + f(x_2)] \quad (6.45)$$

The error of the integral value obtained by using the Simpson's 1/3 rule can be derived in the same manner as that for the trapezoidal rule. Derivation of the error expression is omitted herein so that it will be used as an exercise. The error of the integral value from the Simpson's 1/3 rule is

$$E_t = -\frac{1}{90} h^5 f^{(4)}(\xi) \quad (6.46)$$

By substituting $h = (b-a)/2$ from Eq. (6.44), then

$$E_t = -\frac{(b-a)^5}{2,880} f^{(4)}(\xi) \quad (6.47)$$

where ξ is a value between the integration limit a to b . The error that occurs from the Simpson's 1/3 rule in Eq. (6.47) is less than that in Eq. (6.27) of the trapezoidal rule. Equation (6.47) also implies that the Simpson's 1/3 rule can provide exact integral value if the function to be integrated is a polynomial of third order or lower. Obtaining such exact integral value by using the Simpson's 1/3 rule is demonstrated in the following example.

Example 6.4 Determine the integral in Example 6.1 again but by using the Simpson's 1/3 rule.

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \quad (6.12)$$

Compare the computed integral value with the exact solution of 2.666667.

Herein, the width $h = x_2 - x_1 = x_1 - x_0$ as shown in Eq. (6.44) is

$$h = \frac{b-a}{2} = \frac{2-0}{2} = 1$$

and the values of the function at the three locations are

$$\begin{aligned} f(x_0=0) &= 0 - 0 + 0 + 1 = 1 \\ f(x_1=1) &= 2 - 5 + 3 + 1 = 1 \\ f(x_2=2) &= 16 - 20 + 6 + 1 = 3 \end{aligned}$$

By substituting these values into Eq. (6.43), the approximate integral is

$$I = \frac{1}{3} [1 + 4(1) + 3] = \frac{8}{3} = 2.666667$$

which is equal to the exact solution.

The Simpson's 1/3 rule won't provide exact integral value if the polynomial is of fourth order or higher. For example,

$$I = \int_0^2 (x^4 + 2x^3 - 5x^2 + 3x + 1) dx \quad (6.48)$$

for which the exact solution is 9.066667. In this case, the Simpson's 1/3 rule yields

$$I = \frac{1}{3} [1 + 4(2) + 19] = 9.333333 \quad (6.49)$$

The computed integral has the true error of -0.266667 or -2.9%.

6.5 Composite Simpson's Rule

The Simpson's 1/3 rule can provide more accurate integral value if the integration limits from a to b is divided into many segments. The idea is similar to the composite trapezoidal rule explained in section 6.3. The procedure can be understood clearly by considering Fig. 6.8.

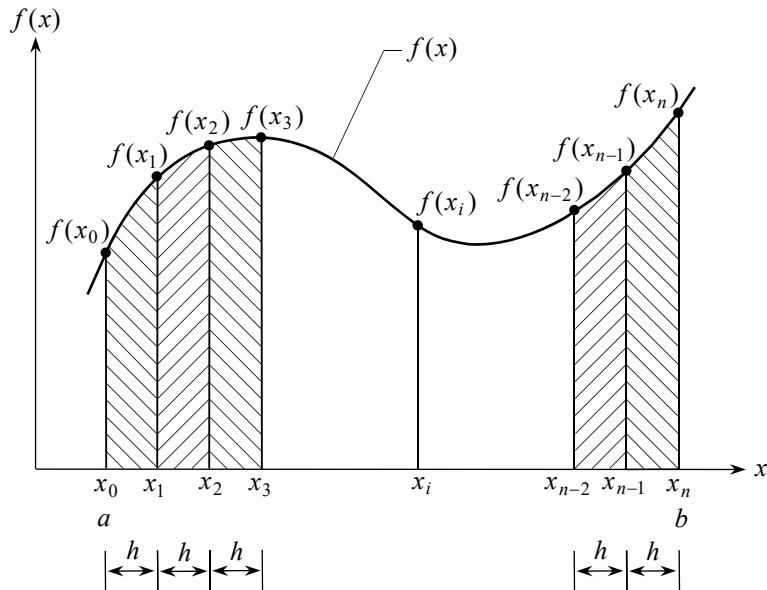


Figure 6.8 Approximate integral determination by composite Simpson's rule.

Figure 6.8 shows distribution of a function $f(x)$ within the interval $a \leq x \leq b$. If the interval from a to b is divided into n segments, then, the width h of each segment is

$$h = \frac{b-a}{n} \quad (6.50)$$

where the coordinates at both ends of each interval are

$$x_i = x_0 + i h \quad i = 0, 1, 2, \dots, n \quad (6.51)$$

Since the general form for the integral of a function $f(x)$ between $a \leq x \leq b$ is

$$I = \int_a^b f(x) dx \quad (6.52)$$

The integral in Eq. (6.52) is firstly divided into $n/2$ sub-integrals with their integration limits between $x_0 \leq x \leq x_2$, $x_2 \leq x \leq x_4$, to $x_{n-2} \leq x \leq x_n$ as

$$I = \int_{x_0}^{x_2} f(x) dx + \int_{x_2}^{x_4} f(x) dx + \dots + \int_{x_{n-2}}^{x_n} f(x) dx \quad (6.53)$$

The Simpson's rule in Eq. (6.43) is then applied to each sub-integral as follow

$$\begin{aligned} I &\approx \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)] + \frac{h}{3} [f(x_2) + 4f(x_3) + f(x_4)] \\ &\quad + \dots + \frac{h}{3} [f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)] \\ &= \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \dots \\ &\quad + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)] \\ I &= \frac{h}{3} \left[f(x_0) + f(x_n) + 4 \sum_{i=1,3,5}^{n-1} f(x_i) + 2 \sum_{i=2,4,6}^{n-2} f(x_i) \right] \end{aligned} \quad (6.54)$$

It should be noted that the number of segments must be an even value because the Simpson's rule is applied to $n/2$ segments. Such constraint must be implemented into the computer program that employs the composite Simpson's rule. The integral error obtained by using the composite Simpson's rule (left as an exercise) is

$$E_a = -\frac{(b-a)^5}{180n^4} \bar{f}^{(4)} \quad (6.55)$$

where $\bar{f}^{(4)}$ is the average value of the fourth-order derivative of the function from all segments. The expression is in the similar form as that for the composite trapezoidal rule as shown by Eq. (6.37).

Example 6.5 Develop a computer program that uses the composite Simpson's rule to estimate the integral

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \quad (6.12)$$

Note that the same integral was previously evaluated in Example 6.1 with the exact solution of 2.666667.

The computer program for determining the integral in this example is presented in Fig. 6.9. Users need to input an even number of the segments n . A warning statement will appear if an odd number of the segments n is input while executing the program. For this example, the program always gives the exact solution of 2.666667 for any input even number n .

```
% Program Simpson
% A multiple-segment Simpson 1/3 program
% for estimating integral of f(x).
func = @(x) ...
    (x^4 + 2*x^3 - 5*x^2 + 3*x + 1);
a = 0; b = 2;
% Read the number of segments required:
n = input( ...
    '\nEnter the number of segments: ');
m = n - fix(n/2)*2;
while m~=0
    disp(' Number of segments must be even')
    return
end
h = (b - a)/n; sum1 = 0.; x = a + h;
for i = 1:2:n-1
    fx = func(x); sum1 = sum1 + fx;
    x = x + 2*h;
end
sum2 = 0; x = a + 2*h;
for i = 2:2:n-2
    fx = func(x); sum2 = sum2 + fx;
    x = x + 2*h;
end
fx0 = func(a); fxn = func(b);
sol = (fx0 + fxn + 4*sum1 + 2*sum2)*h/3;
fprintf( ...
    'The computed integral is %10.6f', sol)
```

Figure 6.9 A computer program to determine the integral in Example 6.5 by using the composite Simpson's rule.

Example 6.6 Modify the computer program developed for Example 6.5 to determine the integral in Eq. (6.48) which is

$$I = \int_0^2 (x^4 + 2x^3 - 5x^2 + 3x + 1) dx \quad (6.48)$$

Investigate the solution improvement by increasing the number of segments n . Compare the computed integral values with the exact solution of 9.066667.

Table 6.2 presents the computed integral values obtained from using different numbers of the segments n . The table shows that the computed integral value approaches the exact solution as the number of the segments n increases.

Table 6.2 Computed integral values by using the composite Simpson's rule with different number of segments n for Example 6.6. The exact integral value is $I = 9.066667$.

<i>n</i>	<i>h</i>	<i>I</i>	ε_t (%)
2	1.0000	9.333333	-2.9412
4	0.5000	9.083333	-.1838
6	0.3333	9.069960	-.0363
8	0.2500	9.067708	-.0115
10	0.2000	9.067093	-.0047
20	0.1000	9.066696	-.0003

6.6 Newton-Cotes Formulas

The trapezoidal rule in section 6.2 estimates the integral value from the area under a straight line or first-order polynomial. Accuracy of the estimated integral value is increased by using the Simpson's 1/3 rule as explained in section 6.4. The Simpson's 1/3 rule estimates the integral value from the area under the second-order polynomial. Thus, a more accurate integral value can be obtained by determining the area under the higher order polynomial.

Figure 6.10 shows the estimation of the integral from the area under the third-order polynomial. Integration of a function $f(x)$ from the limits a to b is

$$I = \int_a^b f(x) dx \quad (6.41)$$

The integral value is approximated by the area under the third-order polynomial as shown by the dashed line in Fig. 6.10. The third-order polynomial can be derived from the general form of the Lagrange polynomial in Eq. (4.32) when $n = 3$. By substituting such third-order polynomial into Eq. (6.41) and performing integration, the approximate integral value is

$$I \approx \frac{3h}{8} [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] \quad (6.56)$$

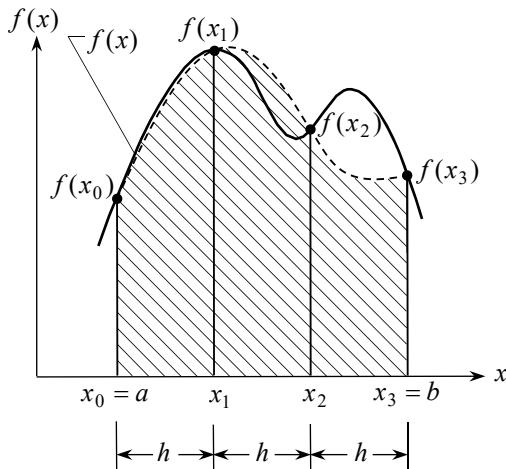


Figure 6.10 Use of the Simpson's 3/8 rule to obtain approximate integral which is the area under the dashed line.

where, in this case

$$h = \frac{b-a}{3} \quad (6.57)$$

Equation (6.56) is called the Simpson's three-eighth rule because the factor of 3/8 appears in front of the expression. It is noted that Eq. (6.56) can be written in a more general form as

$$I \approx \frac{(b-a)}{8} [f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)] \quad (6.58)$$

The integral error from the Simpson's 3/8 rule can be derived in the same manner as that for the trapezoidal rule. The integral error from the Simpson's 3/8 rule is

$$E_t = -\frac{3}{80} h^5 f^{(4)}(\xi) \quad (6.59)$$

By substituting $h = (b-a)/3$ from Eq. (6.57) into Eq. (6.59)

$$E_t = -\frac{(b-a)^5}{6,480} f^{(4)}(\xi) \quad (6.60)$$

where ξ is a value between the integration limits a to b . The integral error as shown in Eq. (6.60) obtained from the Simpson's 3/8 rule is less than that from the Simpson's 1/3 rule in Eq. (6.47). The integral value from the Simpson's 3/8 rule is more accurate because the function is determined at the four locations of x_0 , x_1 , x_2 and x_3 as compared to the three locations used in the Simpson's 1/3 rule.

Example 6.7 Employ the Simpson's 3/8 rule to estimate the integral in Example 6.1

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \quad (6.12)$$

Compare the computed integral value with the exact solution of 2.666667.

The width h to be used in the Simpson's 3/8 rule from Eq. (6.57) for this example is

$$h = \frac{b-a}{3} = \frac{2-0}{3} = \frac{2}{3} \quad (6.61)$$

and the values of the function $f(x)$ at the four locations are

$$\begin{aligned} f(x_0 = 0) &= 0 - 0 + 0 + 1 = 1 \\ f(x_1 = 2/3) &= \frac{16}{27} - \frac{20}{9} + 2 + 1 = \frac{37}{27} \\ f(x_2 = 4/3) &= \frac{128}{27} - \frac{80}{9} + 4 + 1 = \frac{23}{27} \\ f(x_3 = 2) &= 16 - 20 + 6 + 1 = 3 \end{aligned}$$

By substituting these values into Eq. (6.56), the approximate integral obtained from the Simpson's 3/8 rule is

$$I = \left(\frac{3}{8}\right)\left(\frac{2}{3}\right)\left[1 + 3\left(\frac{37}{27}\right) + 3\left(\frac{23}{27}\right) + 3\right] = \frac{8}{3} \quad (6.62)$$

which is equal to the exact solution of 2.666667. The result confirms that the Simpson's 3/8 rule can provide exact integral as implied by Eq. (6.60) if the integrand is a polynomial of third order or less. However, if the integrand is a polynomial of fourth order or higher, the Simpson's 3/8 rule can not provide exact integral as shown in the following example.

Example 6.8 Use the Simpson's 3/8 rule to estimate the integral in Eq. (6.48)

$$I = \int_0^2 (x^4 + 2x^3 - 5x^2 + 3x + 1) dx \quad (6.48)$$

Compare the computed integral value with the exact solution of 9.066667. It is noted that the integral value obtained from the Simpson's 1/3 rule in Eq. (6.49) is 9.333333 with the true error of -2.9%.

The width h according to Eq. (6.57) for the Simpson's 3/8 rule is

$$h = \frac{b-a}{3} = \frac{2-0}{3} = \frac{2}{3} \quad (6.63)$$

and the values of function $f(x)$ at the four locations are

$$\begin{aligned} f(x_0 = 0) &= 0 + 0 - 0 + 0 + 1 = 1 \\ f(x_1 = 2/3) &= \frac{16}{81} + \frac{16}{27} - \frac{20}{9} + 2 + 1 = \frac{127}{81} \\ f(x_2 = 4/3) &= \frac{256}{81} + \frac{128}{27} - \frac{80}{9} + 4 + 1 = \frac{325}{81} \\ f(x_3 = 2) &= 16 + 16 - 20 + 6 + 1 = 19 \end{aligned}$$

By substituting these values into Eq. (6.56) of the Simpson's 3/8 rule, the computed integral solution is

$$I = \left(\frac{3}{8}\right)\left(\frac{2}{3}\right)\left[1 + 3\left(\frac{127}{81}\right) + 3\left(\frac{325}{81}\right) + 19\right] = 9.185185 \quad (6.64)$$

The computed integral solution has the true error from the exact solution of $9.066667 - 9.185185 = -0.118518$ or -1.3%. The error is less than -2.9% which is produced by the Simpson's 1/3 rule.

The trapezoidal rule, Simpson's 1/3 rule and Simpson's 3/8 rule use the first-, second- and third-order Lagrange polynomial, respectively for estimating the integral. Higher order Lagrange polynomials can be used to derive more accurate integral solution. The different orders of the Lagrange polynomials lead to a family of the *Newton-Cotes integration formulas* as follows.

(a) For $n = 1$ with 2 points (trapezoidal rule),

$$I = \frac{b-a}{2}[f(x_0) + f(x_1)] \quad (6.65)$$

The error is $-(1/12)h^3 f^{(2)}(\xi)$ where $h = b - a$.

(b) For $n = 2$ with 3 points (Simpson's 1/3 rule),

$$I = \frac{(b-a)}{6}\left[f(x_0) + 4f(x_1) + f(x_2)\right] \quad (6.66)$$

The error is $-(1/90)h^5 f^{(4)}(\xi)$ where $h = (b-a)/2$.

(c) For $n = 3$ with 4 points (Simpson's 3/8 rule),

$$I = \frac{(b-a)}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right] \quad (6.67)$$

The error is $-(3/80)h^5 f^{(4)}(\xi)$ where $h = (b-a)/3$.

(d) For $n = 4$ with 5 points (Boole's rule),

$$I = \frac{(b-a)}{90}\left[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)\right] \quad (6.68)$$

The error is $-(8/945)h^7 f^{(6)}(\xi)$ where $h = (b-a)/4$.

(e) For $n = 5$ with 6 points,

$$I = \frac{(b-a)}{288} \left[19f(x_0) + 75f(x_1) + 50f(x_2) + 50f(x_3) + 75f(x_4) + 19f(x_5) \right] \quad (6.69)$$

The error is $-(275/12,096) h^7 f^{(6)}(\xi)$ where $h = (b-a)/5$.

(f) For $n = 6$ with 7 points,

$$\begin{aligned} I = & \frac{(b-a)}{840} \left[41f(x_0) + 216f(x_1) + 27f(x_2) + 272f(x_3) + 27f(x_4) \right. \\ & \left. + 216f(x_5) + 41f(x_6) \right] \end{aligned} \quad (6.70)$$

The error is $-(9/1,400) h^9 f^{(8)}(\xi)$ where $h = (b-a)/6$.

(g) For $n = 7$ with 8 points,

$$\begin{aligned} I = & \frac{(b-a)}{17,280} \left[751f(x_0) + 3,577f(x_1) + 1,323f(x_2) + 2,989f(x_3) \right. \\ & \left. + 2,989f(x_4) + 1,323f(x_5) + 3,577f(x_6) + 751f(x_7) \right] \end{aligned} \quad (6.71)$$

The error is $-(8,183/518,400) h^9 f^{(8)}(\xi)$ where $h = (b-a)/7$.

Even though the Newton-Cotes formulas with high numbers of n or points can provide accurate integral solution, they are rarely used in practice. The composite trapezoidal and Simpson's rules are used instead because they are simple. Accurate integral solution can be obtained by employing the composite trapezoidal or Simpson's rule with a large number of segments. The computer programs for the composite trapezoidal and Simpson's rules as shown in Figs. 6.6 and 6.9 can be used to serve for this purpose conveniently.

6.7 Romberg Integration

From the explanation in section 6.3 of the composite trapezoidal rule, the exact integral value I consists of two parts,

$$I = I(h) + E(h) \quad (6.72)$$

The first part is the integral value $I(h)$ obtained from the composite trapezoidal rule such as that shown in Eq. (6.35). The accuracy of the computed integral value depends on the width h of the segments within the integration limits a to b . The second part is the error $E(h)$ as shown in Eq. (6.38)

$$E = -\frac{1}{12} \frac{(b-a)^3}{n^2} \bar{f}'' \quad (6.38)$$

where \bar{f}'' is the average second-derivative from all the segments n as expressed in Eq. (6.37). The error statement in Eq. (6.38) suggests that the integral error E will reduce four times if the number of the segments n is double. Such idea leads the development of the Romberg integration method that can improve the computed integral accuracy. The method divides the integration limits from a to b twice so that the errors obtained from each case are used to produce a more accurate integral solution. The method is based on the *Richardson's extrapolation technique* for which the two integral estimates can lead to a more accurate integral value.

If the composite trapezoidal rule is applied to integrate a given function twice by using the two different widths of h_1 and h_2 , then the exact integral condition in Eq. (6.72) gives

$$I(h_1) + E(h_1) = I(h_2) + E(h_2) \quad (6.73)$$

Since $h = (b - a)/n$, Eq. (6.38) can be written in term of h as

$$E = -\frac{b-a}{12} h^2 \bar{f}'' \quad (6.74)$$

If the average second-derivatives from using the two different widths are assumed to be equal, then the ratio of the errors from the two cases depends on the widths h_1 and h_2 as

$$\frac{E(h_1)}{E(h_2)} = \frac{h_1^2}{h_2^2} \quad (6.75)$$

or,

$$E(h_1) = E(h_2) \left(\frac{h_1}{h_2} \right)^2 \quad (6.76)$$

By substituting Eq. (6.76) into Eq. (6.73),

$$I(h_1) + E(h_2) \left(\frac{h_1}{h_2} \right)^2 = I(h_2) + E(h_2)$$

the error from using the width h_2 can be written in form of the two integral estimates and the two widths as

$$E(h_2) = \frac{I(h_2) - I(h_1)}{\left(h_1/h_2 \right)^2 - 1} \quad (6.77)$$

From Eq. (6.72), the exact integral by using the width h_2 is

$$I = I(h_2) + E(h_2) \quad (6.78)$$

By substituting Eq. (6.77) into Eq. (6.78), a new integral value is obtained from the two integral values that were previously estimated by using the widths h_1 and h_2 as

$$I = I(h_2) + \frac{I(h_2) - I(h_1)}{\left(h_1/h_2 \right)^2 - 1} = \left(1 + \frac{h_2^2}{h_1^2 - h_2^2} \right) I(h_2) - \left(\frac{h_2^2}{h_1^2 - h_2^2} \right) I(h_1) \quad (6.79)$$

Determination of the new integral solution in Eq. (6.79) can be understood clearly if the width h_2 is a half of the width h_1 as

$$h_2 = h_1/2 \quad (6.80)$$

Then, Eq. (6.79) becomes

$$I = \frac{4}{3} I(h_2) - \frac{1}{3} I(h_1) \quad (6.81)$$

Example 6.9 The single and composite trapezoidal rules were used to determine the integral

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \quad (6.12)$$

as shown in Examples 6.1 and 6.3. Their solutions are tabulated as shown below

<i>n</i>	<i>h</i>	<i>I</i>	ε_t (%)
1	2.0	4.00	-50.0
2	1.0	3.00	-12.5
4	0.5	2.75	-3.0

From the table, a new integral value can be determined from two previous integral values when $n = 1$ and $n = 2$ ($h_1 = 2.00$ and $h_2 = 1.00$) according to Eq. (6.81) as

$$I = \frac{4}{3}(3.00) - \frac{1}{3}(4.00) = 2.666667 \quad (6.82)$$

The new integral value obtained in Eq. (6.82) is exact as compared to the exact solution in Eq. (6.14).

Similarly, another new integral value can be obtained from the two previous integral values when $n = 2$ and $n = 4$ ($h_1 = 1.00$ and $h_2 = 0.50$) according to Eq. (6.81) as

$$I = \frac{4}{3}(2.75) - \frac{1}{3}(3.00) = 2.666667 \quad (6.83)$$

In general, the integrand $f(x)$ is not in form of the polynomials, the Romberg integration must be repeated to produce an accurate integral solution. As shown in Eq. (6.74), the error from the trapezoidal rule varies with h^2 or $O(h^2)$. It can be shown that the error is of order $O(h^4)$ after the second application of Romberg integration. The same procedure as shown in Eqs. (6.75) - (6.80) can be repeated so that the new integral solution after the second application of Romberg integration is

$$I = \frac{16}{15} I_M - \frac{1}{15} I_L \quad (6.84)$$

where I_M is the more accurate integral solution and I_L is the less accurate integral solution. The newer integral solution in Eq. (6.84) has the error of order $O(h^6)$. If the Romberg integration technique is applied again, the newer integral solution is

$$I = \frac{64}{63} I_M - \frac{1}{63} I_L \quad (6.85)$$

The new integral solutions obtained from the applications of the Romberg integration as shown in Eqs. (6.81), (6.84), (6.85) can be written in a more general form as

$$I = \frac{2^{2k} I_M - I_L}{2^{2k} - 1} \quad (6.86)$$

where $k = 1, 2, 3, \dots$ represents the k^{th} application of the Romberg integration. Equation (6.86) can be written in another form for convenient computer programming as

$$I = \frac{4^k I_M - I_L}{4^k - 1} \quad (6.87)$$

The example below shows the use of Eq. (6.87) of the Romberg integration method in details.

Example 6.10 Use the Romberg integration method to determine the integral

$$I = \int_0^{\pi/2} \sin x \, dx \quad (6.88)$$

Apply the method until the relative error is less than 0.0001%. Note that the exact solution of the integral value is 1.

The trapezoidal rule is first applied to integrate the function $f(x) = \sin x$ from the limits $a = 0$ to $b = \pi/2$ by using the numbers of intervals $n = 1$ and 2 with the widths of $h_1 = \pi/2$ and $h_2 = \pi/4$, respectively. The applications lead to the two integral values of $I(h_1) = I_L = 0.7853981643$ and $I(h_2) = I_M = 0.9480594490$. The Romberg integration according to Eq. (6.87) is applied with $k = 1$ to give

$$I = \frac{4^1(0.9480594490) - (0.7853981643)}{4^1 - 1} = 1.0022798780 \quad (6.89)$$

The procedure can be shown in a tabulated form for ease of understanding as

<i>n</i>	<i>k = 1</i>	
1	0.7853981643	→ 1.0022798780
2	0.9480594490	↓

The relative error computed from the previous more accurate solution is

$$\varepsilon_a = \left| \frac{1.0022798780 - 0.9480594490}{1.0022798780} \right| \times 100\% = 5.4097\% \quad (6.90)$$

Since the computed error is higher than the specified allowable error, the composite trapezoidal rule is applied again using $n = 4$ with the width $h = \pi/8$. The application yields the corresponding integral value of 0.9871158010. Then, the table becomes

n		$k=1$	$k=2$
1	0.7853981643	1.0022798780	0.9999915655
2	0.9480594490	1.0001345850	
4	0.9871158010		

where the new integral solution after applying the second Romberg integration $k = 2$ is determined from Eq. (6.87) as

$$I = \frac{4^2(1.0001345850) - (1.0022798780)}{4^2 - 1} = 0.9999915655 \quad (6.91)$$

The new integral solution has the relative error of

$$\varepsilon_a = \left| \frac{0.9999915655 - 1.0001345850}{0.9999915655} \right| \times 100\% = 0.0143\% \quad (6.92)$$

which is still higher than the specified allowable error. The composite trapezoidal rule is applied again with $n = 8$ and the width $h = \pi/16$ leading to the integral value of 0.9967851719. The Romberg integration when $k = 3$ is used and the table is updated as

n		$k=1$	$k=2$	$k=3$
1	0.7853981643	1.0022798780	0.9999915655	1.0000000090
2	0.9480594490	1.0001345850	0.9999998771	
4	0.9871158010	1.0000082960		
8	0.9967851719			

where the newer integral solution of 1.0000000090 is determined from Eq. (6.87) when $k = 3$ as

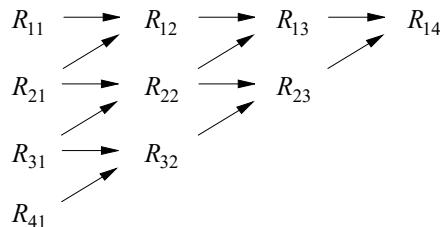
$$I = \frac{4^3(0.9999998771) - (0.9999915655)}{4^3 - 1} = 1.0000000090 \quad (6.93)$$

With the newer integral solution, the relative error is

$$\varepsilon_a = \left| \frac{1.0000000090 - 0.9999998771}{1.0000000090} \right| \times 100\% = 0.00001\% \quad (6.94)$$

Since the relative error is now less than the specified allowable error of 0.0001%, the Romberg integration process is terminated with the final integral solution of 1.0000000090.

The Romberg integration process as explained in Example 6.10 can be used to develop a computer program such as that shown in Fig. 6.11. The program can be used to integrate an arbitrary function $f(x)$ from the given integration limits a to b and the specified relative error. It is noted that the program stores the integral values in the format as shown below.



In the above format, R_{14} is the final integral solution.

```

% Program Romberg
% A Romberg integration program for
% estimating integral of f(x) within
% a specified error.
func = @(x)(sin(x));
a = 0.; b = pi/2.; es = 0.0001;
% Compute r(1,1):
fx0 = func(a); fxn = func(b);
r(1,1) = (fx0 + fxn)*(b - a)/2.;
% Loop over number of Romberg application:
for i = 1:9
  n = 2^i; area = Trap(a, b, n);
  r(i+1,1) = area;
  for ic = 2:i+1
    k = ic - 1; ir = 2 + i - ic;
    r(ir,ic) = ((4^k)*r(ir+1,k) ...
    - r(ir,k))/(4^k - 1);
  end
  err = 100.* (r(1,k+1) - r(2,k))/r(1,k+1);
  err = abs(err);
end
if err <= es
  continue
end
fprintf( ...
  '\nFinal integral value = %16.10e', ...
  r(1,k+1));
fprintf( ...
  '\nwith relative error = %12.6e', ...
  err);


---


function area = Trap(a, b, n)
% Multiple-segment trapezoidal rule.
func = @(x)(sin(x));
h = (b - a)/n; sum = 0.; x = a + h;
for i = 1:n-1
  fx = func(x); sum = sum + fx; x = x + h;
end
fx0 = func(a); fxn = func(b);
area = (fx0 + fxn + 2.*sum)*h/2.;
  
```

Figure 6.11 Computer program for determining the integral solution in Example 6.10 by using the Romberg integration method.

6.8 Gauss Integration

Gauss integration is one of the most popular integration methods widely used in science and engineering computation. The method is sometimes called Gauss quadrature because the integral is determined from the approximate quadrilateral area under the function to be integrated. Determining the quadrilateral area by using the Gauss integration method is similar to that for the trapezoidal rule as explained in section 6.2. However, the quadrilateral area obtained from the Gauss integration method is more accurate in representing the integral as depicted in Fig. 6.12.

Figure 6.12 shows the difference in the integral estimation from the area under the function $f(x)$ by using the trapezoidal rule and Gauss quadrature. If the distribution of the function $f(x)$ to be integrated is in the concave form as shown in Fig. 6.12(a), the trapezoidal rule will produce a considerable error because the quadrilateral area is determined from the two values of the function $f(x)$ evaluated at the

integration limits a and b . The error is substantial if the integrating interval from a to b is large. Figure 6.12(b) shows that a more accurate integral solution can be obtained if the quadrilateral area is determined from the two values of the function $f(x)$ at the two locations away from the two integration limits a and b . The question is, thus, what should be the two locations that can provide a more accurate integral solution.

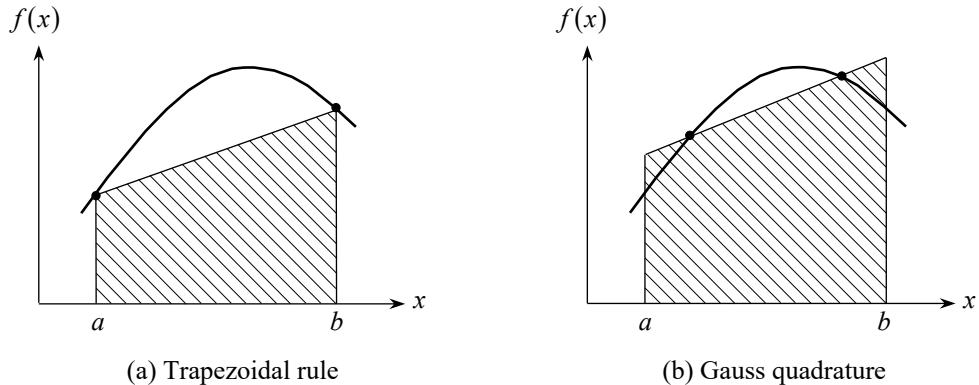


Figure 6.12 Integral determination from the quadrilateral area by the trapezoidal rule and Gauss quadrature.

It is noted that the Newton-Cotes formulas as shown in Eqs. (6.65) - (6.71) can be written in a general form as

$$I = \int_a^b f(x) dx \cong \sum_{i=1}^n W_i f(x_i) \quad (6.95)$$

For example, the integral formula for the trapezoidal rule in Eq. (6.65) can be written in the form

$$I = W_1 f(x_1) + W_2 f(x_2) \quad (6.96)$$

where W_1 and W_2 may be thought as the weights that correspond to the functions evaluated at two appropriate locations. In this case,

$$W_1 = W_2 = \frac{b-a}{2} \quad (6.97a)$$

$$f(x_1) = f(a) \quad ; \quad f(x_2) = f(b) \quad (6.97b)$$

The integral formula in Eq. (6.96) with the weights and locations in Eqs. (6.97a-b) produce the quadrilateral area as shown in Fig. 6.12(a).

To develop the Gauss integration formula, the integral form as shown in Eq. (6.96) is used. However, the integration limits are from -1 to $+1$ in the ξ -coordinate direction as shown in Fig. 6.13 so that the developed formula can be used in general. Thus, the integral statement is written in the form

$$\begin{aligned}
 I &= \int_{-1}^1 f(\xi) d\xi \\
 &= W_1 f(\xi_1) + W_2 f(\xi_2)
 \end{aligned} \tag{6.98}$$

where the weights W_1 , W_2 and the locations ξ_1 , ξ_2 are unknowns.

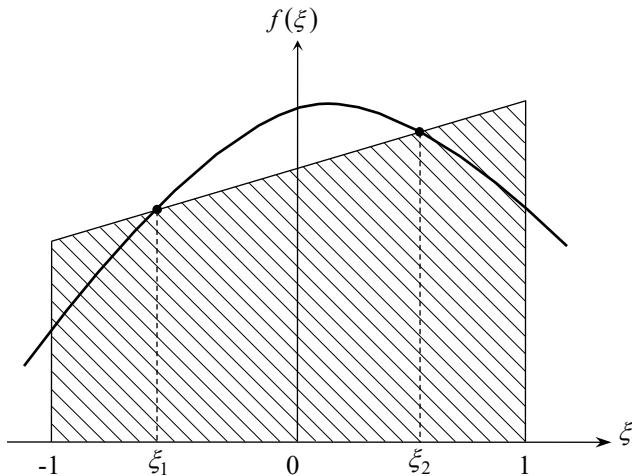


Figure 6.13 Gauss integration in the ξ -coordinate direction.

The four unknowns are to be determined from the four appropriate conditions. The four conditions are such that the exact integral must be obtained by using Eq. (6.98) if the function $f(\xi)$ is constant, linear, parabola or cubic, i.e.,

$$f(\xi) = 1, \xi, \xi^2, \xi^3 \tag{6.99}$$

The function $f(\xi)$ in the four forms of Eq. (6.99) lead to the four conditions according to Eq. (6.98) as follows,

$$W_1 + W_2 = \int_{-1}^1 1 d\xi = 2 \tag{6.100a}$$

$$W_1 \xi_1 + W_2 \xi_2 = \int_{-1}^1 \xi d\xi = 0 \tag{6.100b}$$

$$W_1 \xi_1^2 + W_2 \xi_2^2 = \int_{-1}^1 \xi^2 d\xi = \frac{2}{3} \tag{6.100c}$$

$$W_1 \xi_1^3 + W_2 \xi_2^3 = \int_{-1}^1 \xi^3 d\xi = 0 \tag{6.100d}$$

Eqs. (6.100a-d) are nonlinear with the four unknowns of W_1 , W_2 , ξ_1 and ξ_2 . These unknowns can be determined by starting from Eq. (6.100b)

$$W_2 = -\frac{W_1 \xi_1}{\xi_2} \quad (6.101a)$$

By substituting W_2 from Eq. (6.101a) into Eq. (6.100d),

$$W_1 \xi_1^3 - W_1 \xi_1 \xi_2^2 = 0$$

or,

$$\xi_1^2 = \xi_2^2$$

But ξ_1 is not equal to ξ_2 , then

$$\xi_1 = -\xi_2 \quad (6.101b)$$

By substituting ξ_1 into Eq. (6.101a), the weight $W_1 = W_2$. With the use of Eq. (6.100a), it is found that

$$W_1 = W_2 = 1 \quad (6.101c)$$

Then, by substituting Eqs. (6.101b-c) into Eq. (6.100c),

$$\xi_1^2 + \xi_2^2 = \frac{2}{3}$$

to give

$$\xi_1 = -\frac{1}{\sqrt{3}} \quad (6.101d)$$

In summary, the weights W_1 , W_2 and the two locations ξ_1 , ξ_2 are

$$W_1 = W_2 = 1 \quad (6.102a)$$

$$\xi_1 = -\frac{1}{\sqrt{3}} = -0.5773502692 \quad (6.102b)$$

$$\xi_2 = +\frac{1}{\sqrt{3}} = +0.5773502692 \quad (6.102c)$$

The locations ξ_1 and ξ_2 are called the Gauss point locations. With the weights and locations in Eqs. (6.102a-c), the integral formula for the two-point Gauss integration in Eq. (6.98) is

$$I = (1)f\left(-\frac{1}{\sqrt{3}}\right) + (1)f\left(+\frac{1}{\sqrt{3}}\right) \quad (6.103)$$

The formula provides an exact integral solution if the function to be integrated is a polynomial of third order or less.

If the function to be integrated is a higher order polynomial or other complicated functions, the same procedure as shown in Eqs. (6.98) - (6.103) can still be applied to derive the Gauss integration formulas to produce more accurate integral solutions. For example, a more accurate integral solution can be derived by using 3 terms in the formula as

$$\begin{aligned} I &= \int_{-1}^{1} f(\xi) d\xi \\ &= W_1 f(\xi_1) + W_2 f(\xi_2) + W_3 f(\xi_3) \end{aligned} \quad (6.104)$$

The six unknowns in Eq. (6.104) can be determined to yield

$$W_1 = W_3 = 5/9 \quad ; \quad W_2 = 8/9 \quad (6.105a)$$

$$\xi_1 = -\sqrt{0.6} = -0.7745966692 \quad ; \quad \xi_2 = 0 \quad (6.105b)$$

$$\xi_3 = +\sqrt{0.6} = +0.7745966692 \quad (6.105c)$$

So that the integral formula for the three-point Gauss integration is

$$I = \left(\frac{5}{9}\right)f(-\sqrt{0.6}) + \left(\frac{8}{9}\right)f(0) + \left(\frac{5}{9}\right)f(+\sqrt{0.6}) \quad (6.106)$$

which provides an exact integral solution if the function to be integrated is a polynomial of fifth order or less. The same procedure above can be further applied to derive the Gauss integration formulas with the weights W_i and locations ξ_i for n Gauss points as shown in Table 6.3. The formulas are known as the *Gauss-Legendre formulas* for which the integral value is determined from

$$I = \int_{-1}^1 f(\xi) d\xi \cong \sum_{i=1}^n W_i f(\xi_i) \quad (6.107)$$

From the explanation above, the Gauss integration formulas can produce an exact integral solution if the function to be integrated is a polynomial of order $2n-1$ or less.

For n Gauss points, the weights W_i and locations ξ_i as shown in Table 6.3 are determined by first assuming the function $f(\xi)$ in the form

$$f(\xi) = 1, \xi, \xi^2, \xi^3, \dots, \xi^{2n-1} \quad (6.108)$$

Then, the same procedure as explained by Eqs. (6.99)-(6.100) for the two-point integration is applied to yield $2n$ equations with $2n$ unknowns as follows,

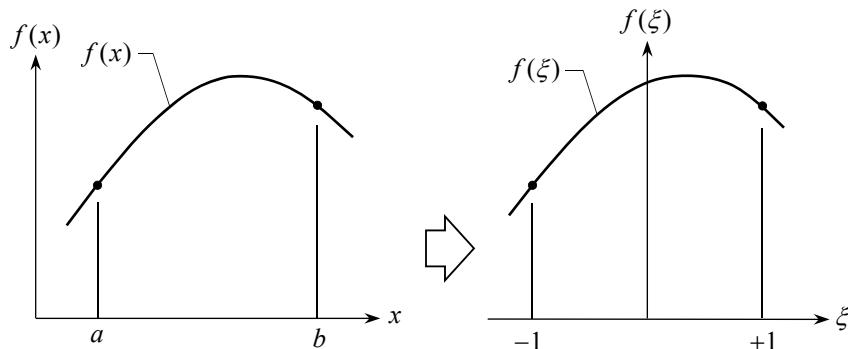
$$\begin{aligned} W_1 + W_2 + \dots + W_n &= 2 \\ W_1 \xi_1 + W_2 \xi_2 + \dots + W_n \xi_n &= 0 \\ W_1 \xi_1^2 + W_2 \xi_2^2 + \dots + W_n \xi_n^2 &= \frac{2}{3} \\ W_1 \xi_1^3 + W_2 \xi_2^3 + \dots + W_n \xi_n^3 &= 0 \\ &\vdots \\ W_1 \xi_1^{2n-2} + \dots + W_n \xi_n^{2n-2} &= \frac{2}{2n-1} \\ W_1 \xi_1^{2n-1} + \dots + W_n \xi_n^{2n-1} &= 0 \end{aligned} \quad (6.109)$$

The values of the weights W_i and locations ξ_i as shown in Table 6.3 are obtained by solving Eq. (6.109).

Table 6.3 Weights and locations of the Gauss-Legendre formulas.

Gauss points	Locations	Weights
<i>n</i>	$\pm \xi_i$	W_i
1	0.0000000000	2.0000000000
2	0.5773502692	1.0000000000
3	0.0000000000	0.8888888889
	0.7745966692	0.5555555556
4	0.3399810436	0.6521451549
	0.8611363116	0.3478548451
5	0.0000000000	0.5688888889
	0.5384693101	0.4786286705
	0.9061798459	0.2369268850
6	0.2386191861	0.4679139346
	0.6612093865	0.3607615730
	0.9324695142	0.1713244924

The Gauss-Legendre formulas in Eq. (6.107) were developed for integrating a function $f(\xi)$ from the limits -1 to +1 along the ξ -coordinate. Since a given integrand is normally in form of the function $f(x)$ that needs to be integrated along the x -coordinate from the lower limit a to upper limit b , the coordinate transformation from x - to ξ -coordinate must be first performed. Transformation of the function $f(x)$ from x - to ξ -coordinate can be done easily by considering Fig. 6.14.

**Figure 6.14** Transformation of the function to be integrated from x - to ξ -coordinate before applying Gauss-Legendre integration formulas.

Coordinate transformation from x to ξ may be done by using linear relation in the form

$$x = c_0 + c_1 \xi \quad (6.110)$$

where c_0 and c_1 are constants that can be determined from the integration limits as follows

$$\text{at } x=a \quad ; \quad a = c_0 + c_1(-1) \quad (6.111a)$$

$$\text{at } x=b \quad ; \quad b = c_0 + c_1(+1) \quad (6.111b)$$

which give

$$c_0 = \frac{a+b}{2} \quad \text{and} \quad c_1 = \frac{b-a}{2} \quad (6.112)$$

Then, the relation between the two coordinates is

$$x = \frac{a+b}{2} + \frac{b-a}{2}\xi \quad (6.113)$$

And

$$dx = \frac{b-a}{2}d\xi \quad (6.114)$$

In summary, the Gauss integration starts from the integral of a function $f(x)$ along the x -coordinate with the limits of a and b as

$$I = \int_a^b f(x)dx \quad (6.115)$$

The integral is first transformed from x - to ξ -coordinate by using Eqs. (6.113) and (6.114) so that the integration limits change from a and b to -1 and +1 as

$$\begin{aligned} I &= \int_{-1}^1 f\left(\frac{a+b}{2} + \frac{b-a}{2}\xi\right)\left(\frac{b-a}{2}\right)d\xi \\ &= \left(\frac{b-a}{2}\right) \int_{-1}^1 f\left(\frac{a+b}{2} + \frac{b-a}{2}\xi\right)d\xi \end{aligned} \quad (6.116)$$

Then, the Gauss-Legendre formula according to Eq. (6.107) is applied to yield

$$I = \left(\frac{b-a}{2}\right) \sum_{i=1}^n W_i f\left(\frac{a+b}{2} + \frac{b-a}{2}\xi_i\right) \quad (6.117)$$

The entire process of the Guass integration method can be understood clearly by solving the integral in Example 6.1 again as shown in the following example.

Example 6.11 Use the two-point Gauss integration method to integrate

$$I = \int_0^2 f(x)dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1)dx \quad (6.12)$$

Then, develop a computer program for integrating a given function by using one- to six-point Gauss formulas. Compare the computed integrals obtained from each case with the exact solution of 2.666667.

By starting from the coordinate transformation from x to ξ according to Eq. (6.113)

$$x = \frac{a+b}{2} + \frac{b-a}{2}\xi \quad (6.113)$$

Since the integration limits are $a = 0$ and $b = 2$, Eq. (6.113) reduces to

$$x = 1 + \xi \quad (6.118)$$

Then, the integral Eq. (6.12) becomes

$$\begin{aligned}
 I &= \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \\
 &= \int_{-1}^1 \left[2(1+\xi)^3 - 5(1+\xi)^2 + 3(1+\xi) + 1 \right] \left(\frac{2-0}{2} \right) d\xi
 \end{aligned}$$

By using the two-point Gauss formula with $n = 2$, Eq. (6.117) is

$$\begin{aligned}
 I &= \left(\frac{2-0}{2} \right) \sum_{i=1}^2 W_i \left[2(1+\xi_i)^3 - 5(1+\xi_i)^2 + 3(1+\xi_i) + 1 \right] \\
 &= W_1 \left[2(1+\xi_1)^3 - 5(1+\xi_1)^2 + 3(1+\xi_1) + 1 \right] \\
 &\quad + W_2 \left[2(1+\xi_2)^3 - 5(1+\xi_2)^2 + 3(1+\xi_2) + 1 \right]
 \end{aligned}$$

But from Table 6.3, $W_1 = W_2 = 1$ and $\xi_1 = -\sqrt{1/3}$, $\xi_2 = +\sqrt{1/3}$, then

$$\begin{aligned}
 I &= (1)[0.1509982 - 0.8931639 + 1.2679492 + 1] \\
 &\quad + (1)[7.8490018 - 12.440169 + 4.7320508 + 1] \\
 I &= 2.666667
 \end{aligned}$$

which is equal to the exact solution because the two-point Gauss formula can provide exact integral if the integrand is a polynomial of third order or less.

Figure 6.15 shows a computer program for the Gauss-Legendre integration method by using one to six integration points. The program was developed so that the computational procedure is easy to understand. The program starts from declaring the values of the Gauss locations ξ and their weights W . The program reads the integration limits a and b from user input. The function $f(x)$ to be integrated is declared in the program. The function can be replaced by the new function desired for integration.

```

% Program GausInt
% A Gauss-Legendre integration program for
% estimating integral of function f(x) by
% using 1 through Gauss points.
func = @(x)(2.*x^3 - 5.*x^2 + 3.*x + 1.);
xi = [ 0.000000; -0.5773503; 0.5773503;
       -0.7745967; 0.000000; 0.7745967;
       -0.8611363; -0.3399810; 0.3399810;
       0.8611363; -0.9061798; -0.5384693;
       0.000000; 0.5384693; 0.9061798;
       -0.9324695; -0.6612094; -0.2386192;
       0.2386192; 0.6612094; 0.9324695];
w = [ 2.000000; 1.000000; 1.000000;
       0.5555556; 0.8888889; 0.5555556;
       0.3478549; 0.6521452; 0.6521452;
       0.3478549; 0.2369269; 0.4786287;
       0.5688889; 0.4786287; 0.2369269;
       0.1713245; 0.3607616; 0.4679139;
       0.4679139; 0.3607616; 0.1713245];
% Read limits for integration:
a = input('Enter value of a: ');
b = input('Enter value of b: ');
a0 = (a + b)/2.;
a1 = (b - a)/2.;
% Perform numerical integration by using
% 1 through 6 Gauss points:
ic = 1;
for ng = 1:6
    sum = 0.;
    for iterms = 1:ng
        x = a0 + a1*xi(ic); ai = func(x);
        sum = sum + w(ic)*ai; ic = ic + 1;
    end
    sum = a1*sum;
    fprintf( ...
        '\nResult of integration with %ld', ng);
    fprintf( ...
        ' Gauss point(s) is %12.6e', sum);
end

```

Figure 6.15 Computer program for Gauss integration method.

Figure 6.16 shows the computed integral values of Example 6.11 obtained from using the computer program in Fig. 6.15. The exact integral solution is 2.666667. The figure shows that the computed integral values are exact when two- to six-point Gauss integration formulas are used since the function to be integrated is a third-order polynomial. It should be noted that for a general function $f(x)$ that may be complicated, more accurate solutions are obtained by using the increased number of Gauss integration points.

```
Result of integration wiht 1 Gauss point(s) is .200000E+01
Result of integration wiht 2 Gauss point(s) is .266667E+01
Result of integration wiht 3 Gauss point(s) is .266667E+01
Result of integration wiht 4 Gauss point(s) is .266667E+01
Result of integration wiht 5 Gauss point(s) is .266667E+01
Result of integration wiht 6 Gauss point(s) is .266667E+01
```

Figure 6.16 Computed integral values of Example 6.11 from the Gauss integration computer program in Fig. 6.15.

6.9 Multiple Integration

Multiple integrations that are frequently used in science and engineering are the double integration over the area A

$$I = \iint_A f(x, y) dA = \iint_A f(x, y) dx dy \quad (6.119)$$

and the triple integration over the volume V

$$I = \iiint_V f(x, y) dV = \iiint_V f(x, y) dx dy dz \quad (6.120)$$

Multiple integrations can be done straightforwardly by performing the single integration in each coordinate direction, one at a time.

For example, the double integration of a function $f(x, y)$ in Eq. (6.119) for the intervals $a < x < b$ and $c < y < d$ can be expressed in the form

$$I = \iint_A f(x, y) dx dy = \int_a^b \left(\int_c^d f(x, y) dy \right) dx \quad (6.121)$$

The function $f(x, y)$ is first integrated along the y -coordinate direction by keeping x constant. The result is integrated again along the x -coordinate direction to obtain the final integral solution.

The double integral of the function $f(x, y)$ in Eq. (6.121) can also be performed by first integrating the function along the x -coordinate direction in the inner bracket as shown below

$$I = \int_c^d \left(\int_a^b f(x, y) dx \right) dy \quad (6.122)$$

A numerical method in the form of Eq. (6.95) can be applied to approximate the integral in the inner bracket along the x -direction for Eq. (6.122) as

$$I \equiv \int_c^d \left(\sum_{i=1}^n W_{xi} f(x_i, y) \right) dy \quad (6.123)$$

For an example of using the trapezoidal rule ($n = 2$), the weights are $W_{x1} = W_{x2} = (b - a)/2$. The method is applied again to numerically integrate the result obtained along the y -coordinate direction to yield

$$I \equiv \sum_{i=1}^n \sum_{j=1}^n W_{xi} W_{yj} f(x_i, y_j) \quad (6.124)$$

where $W_{y1} = W_{y2} = (d - c)/2$. Other numerical methods, such as the Simpson's rule and the Boole's rule, can be applied in the same fashion.

Example 6.12 Use the trapezoidal rule to determine the double integral

$$I = \int_0^1 \int_1^2 (1+x)y \, dx \, dy \quad (6.125)$$

Herein, $a = 1$, $b = 2$, $c = 0$, $d = 1$, $f(x, y) = (1+x)y$ and $W_{x1} = W_{x2} = (b - a)/2 = 0.5$, $W_{y1} = W_{y2} = (d - c)/2 = 0.5$, then, Eq. (6.124) yields

$$\begin{aligned} I &= W_{x1} W_{y1} f(x_1, y_1) + W_{x2} W_{y1} f(x_2, y_1) + W_{x1} W_{y2} f(x_1, y_2) + W_{x2} W_{y2} f(x_2, y_2) \\ &= W_{x1} W_{y1} f(a, c) + W_{x2} W_{y1} f(b, c) + W_{x1} W_{y2} f(a, d) + W_{x2} W_{y2} f(b, d) \\ &= (0.5)(0.5)f(1, 0) + (0.5)(0.5)f(2, 0) + (0.5)(0.5)f(1, 1) + (0.5)(0.5)f(2, 1) \\ &= (0.25)(0) + (0.25)(0) + (0.25)(2) + (0.25)(3) \\ &= 0 + 0 + 0.5 + 0.75 \\ I &= 1.25 \end{aligned} \quad (6.126)$$

The computed integral solution is exact because the given function $f(x, y)$ is linear in both x - and y -directions. The method won't provide exact solution if the function $f(x, y)$ is in a more complex form, such as $f(x, y) = (1+x^2)y^3$ or $f(x, y) = (1+\sin x)e^y$. However, accurate integral solution of these functions can be obtained by using composite trapezoidal rule with many segments.

In solving practical science and engineering problems, the Gauss integration method may be preferred because of its solution accuracy. To apply the Gauss integration method for the double integration, the integral limits from a to b in the x -direction are first changed to -1 and $+1$ in the ξ -direction. Similarly, the integral limits from c to d in the y -direction are first changed to -1 and $+1$ in the η -direction. The linear relations between the x - y and ξ - η coordinate systems similar to that explained in Eq. (6.110) and Example 6.11 can be used for both the x - and y -directions as

$$x = \frac{a+b}{2} + \frac{b-a}{2}\xi \quad ; \quad dx = \frac{b-a}{2}d\xi \quad (6.127a)$$

$$y = \frac{c+d}{2} + \frac{d-c}{2}\eta \quad ; \quad dy = \frac{d-c}{2}d\eta \quad (6.127b)$$

Then, the double integral in Eq. (6.122) becomes

$$\begin{aligned}
 I &= \int_c^d \left(\int_a^b f(x, y) dx \right) dy \\
 &= \int_c^d \left(\int_{-1}^1 f\left(\frac{a+b}{2} + \frac{b-a}{2}\xi, y\right) \left(\frac{b-a}{2}\right) d\xi \right) dy \\
 &= \int_c^d \left(\left(\frac{b-a}{2}\right) \int_{-1}^1 f\left(\frac{a+b}{2} + \frac{b-a}{2}\xi, y\right) d\xi \right) dy \\
 &= \left(\frac{b-a}{2}\right) \left(\frac{d-c}{2}\right) \int_{-1}^1 \int_{-1}^1 f\left(\frac{a+b}{2} + \frac{b-a}{2}\xi, \frac{c+d}{2} + \frac{d-c}{2}\eta\right) d\xi d\eta \\
 &= \left(\frac{b-a}{2}\right) \left(\frac{d-c}{2}\right) \sum_{i=1}^n \sum_{j=1}^n W_i W_j f\left(\frac{a+b}{2} + \frac{b-a}{2}\xi_i, \frac{c+d}{2} + \frac{d-c}{2}\eta_j\right)
 \end{aligned} \tag{6.128}$$

For example, the two Gauss points ($n = 2$) in each ξ - and η -coordinate direction is used, the weights are $W_1 = W_2 = 1$ with the Gauss point locations of $\xi_1 = \eta_1 = -\sqrt{1/3}$ and $\xi_2 = \eta_2 = +\sqrt{1/3}$.

Example 6.13 Use the Gauss integration method to determine the double integral

$$I = \int_0^1 \int_1^2 (1+x)y dx dy \tag{6.125}$$

Herein, $a = 1$, $b = 2$, $c = 0$, $d = 1$, then, Eqs. (6.127a-b) are

$$\begin{aligned}
 x &= 1.5 + 0.5\xi & ; & \quad dx = 0.5d\xi \\
 y &= 0.5 + 0.5\eta & ; & \quad dy = 0.5d\eta
 \end{aligned}$$

Then, Eq. (6.125) becomes

$$\begin{aligned}
 I &= \int_{-1}^1 \int_{-1}^1 [(1+1.5+0.5\xi)(0.5+0.5\eta)](0.5d\xi)(0.5d\eta) \\
 &= (0.25) \int_{-1}^1 \int_{-1}^1 [(2.5+0.5\xi)(0.5+0.5\eta)] d\xi d\eta \\
 &= (0.0625) \int_{-1}^1 \int_{-1}^1 (5+\xi)(1+\eta) d\xi d\eta
 \end{aligned}$$

By using the two Gauss points ($n=2$) in each ξ - and η -coordinate direction,

$$\begin{aligned}
 I &= (0.0625) \sum_{i=1}^2 \sum_{j=1}^2 W_i W_j (5+\xi_i)(1+\eta_j) \\
 &= (0.0625)[W_1 W_1 (5+\xi_1)(1+\eta_1) + W_1 W_2 (5+\xi_1)(1+\eta_2) \\
 &\quad + W_2 W_1 (5+\xi_2)(1+\eta_1) + W_2 W_2 (5+\xi_2)(1+\eta_2)]
 \end{aligned}$$

With the weights of $W_1 = W_2 = 1$, and the Gauss point locations of $\xi_1 = \eta_1 = -\sqrt{1/3}$, $\xi_2 = \eta_2 = +\sqrt{1/3}$, then

$$\begin{aligned} I &= (0.0625) \left[(1)(1)(5 - \sqrt{1/3})(1 - \sqrt{1/3}) + (1)(1)(5 - \sqrt{1/3})(1 + \sqrt{1/3}) \right. \\ &\quad \left. + (1)(1)(5 + \sqrt{1/3})(1 - \sqrt{1/3}) + (1)(1)(5 + \sqrt{1/3})(1 + \sqrt{1/3}) \right] \\ &= (0.0625)[1.869232 + 6.976068 + 2.357266 + 8.797435] \\ &= (0.0625)(20) \\ I &= 1.25 \end{aligned}$$

The Gauss integration method yields the exact integral solution because the given function $f(x, y)$ is linear in both x - and y -directions. If the given function $f(x, y)$ is complicated, many Gauss points must be used to produce a more accurate solution. Since the Gauss integration method provides high integral solution accuracy, the method is widely used for solving practical problems by embedding itself in many commercial software.

6.10 MATLAB Commands for Integration

One of the simple MATLAB commands for integration is `trapz`. The command employs the composite trapezoidal rule to estimate the integral value of a given function. The format for using the command is

$$z = \text{trapz}(x, y)$$

- where x is the data along the x -coordinate
- y is the function to be integrated
- z is the integral value

Example 6.14 Determine the integral in Eq. (6.12) by using the MATLAB command `trapz`.

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx \quad (6.12)$$

Compare the computed value with the exact solution of 2.666667.

It is noted that the command `trapz` implicitly specifies the segment length h in the x input data. As an example of $h = 0.4$, the input data, function and output in x , y and z variables are as follows,

```
>> x = [0:0.4:2];
>> y = 2*x.^3-5*x.^2+3*x+1;
>> z = trapz(x,y)
```

```
Z =
2.7200
```

If the segment length h is reduced to 0.01, the output solution is closer to the exact solution.

```
>> x = [0:0.01:2];
>> y = 2*x.^3-5*x.^2+3*x+1;
>> z = trapz(x,y)

z =
2.6667
```

Another useful MATLAB command for integration is `cumtrapz`. The command determines cumulative integral. The format of the command is

$$z = \text{cumtrapz}(x, y)$$

where
 x is the data along the x -coordinate
 y is the function to be integrated
 z is the cumulative integral value. For example, $z(3)$ is the integral value for the interval from $x(1)$ to $x(3)$.

MATLAB also provides simple commands for double and triple integrations. The command format for double integration is,

$$I = \text{dblquad}(\text{function}, \text{xmin}, \text{xmax}, \text{ymin}, \text{ymax}, \text{tol})$$

where
 I is the integral value
 function is the function to be integrated input by using the command `inline`
 xmin is the lower limit of x
 xmax is the upper limit of x
 ymin is the lower limit of y
 ymax is the upper limit of y
 tol is the specified tolerance, default value is 1×10^{-6} .

Example 6.15 Use MATLAB function `dblquad` to determine the integral

$$I = \int_0^1 \int_1^2 (1+x)y \, dx \, dy \quad (6.125)$$

The command entered and the output integral value are as follows.

```
>> I = dblquad(inline(' (1+x)*y', 'x', 'y'), 1, 2, 0, 1)

I =
1.2500
```

The MATLAB command `triplequad` is used for determining the triple integration. The command format is similar to the `dblquad` command except the additional limits in the z -coordinate,

$$I = \text{triplequad}(\text{function}, \text{xmin}, \text{xmax}, \text{ymin}, \text{ymax}, \text{zmin}, \text{zmax}, \text{tol})$$

Example 6.16 Use the MATLAB command `triplequad` to determine the integral

$$\int_{-2}^2 \int_0^2 \int_{-3}^1 (x^3 - 3yz) dx dy dz \quad (6.129)$$

It is noted that the exact solution is -160. The command entered and the output integral value from MATLAB are as follows.

```
>> I = triplequad(inline('x.^3-3*y*z', 'x', 'y', 'z'), -3, 1, 0, 2, -2, 2)
I =
-160.0000
```

6.11 Differentiation

Understanding physical meanings of derivatives and procedures for determining them are essential to study higher-level numerical methods in the following chapters. The meaning for the derivative of a function $f(x)$ is explained by Fig. 6.17.

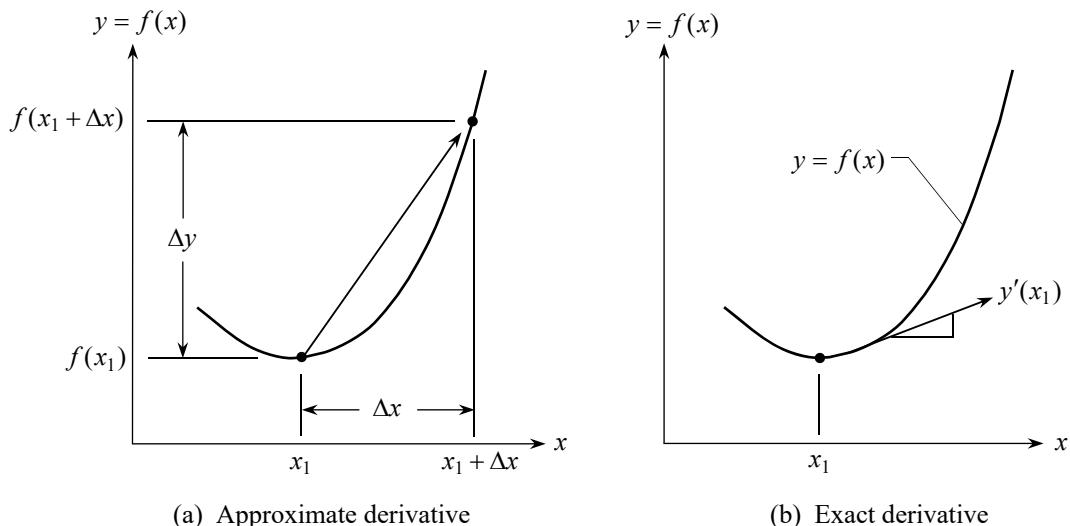


Figure 6.17 Approximate and exact derivatives.

Figure 6.17(a) shows the distribution of a function y that varies with the independent variable x . An approximate derivative of the function y with respect to x is

$$\frac{\Delta y}{\Delta x} = \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x} \quad (6.130)$$

As Δx approaches zero, the exact derivative at x_1 is

$$\frac{dy}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x} \quad (6.131)$$

The exact derivative dy/dx is the slope at x_1 normally denoted by y' or $f'(x)$ as shown in Fig. 6.17(b).

Many functions $f(x)$ studied in high school or first year college are simple and their derivatives can be determined exactly. For example,

$$y = f(x) = x^n \quad (6.132)$$

then, its exact derivative is

$$\frac{dy}{dx} = \frac{df(x)}{dx} = nx^{n-1} \quad (6.133)$$

Most of the functions in practice, however, are complicated and their exact derivatives can not be determined easily. Thus, their approximate derivatives are determined instead by starting from the Taylor's series as shown in Eq. (2.29),

$$f(x_{i+1}) = f(x_i) + h f'(x_i) + \frac{h^2}{2!} f''(x_i) + \dots \quad (6.134)$$

where h is the width between the locations x_i and x_{i+1} . From Eq. (6.134), the first derivative of the function $f(x)$ at x_i is

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2!} f''(x_i) + \dots \quad (6.135)$$

$$\text{Or, } f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h) \quad (6.136)$$

where $O(h)$ represents the error of order h if the first term on the right-hand side of Eq. (6.136) is used to approximate the derivative. The approximation is sometimes called the *first forward divided difference* because the two values of the function $f(x)$ at x_i and x_{i+1} are used to determine the derivative as shown in Fig. 6.18(a).

Similar to Eq. (6.134), the Taylor's series can be used to determine the value of the function $f(x)$ at location x_{i-1} as

$$f(x_{i-1}) = f(x_i) - h f'(x_i) + \frac{h^2}{2!} f''(x_i) - \dots \quad (6.137)$$

So that the first-order derivative at x_i is

$$f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + \frac{h}{2!} f''(x_i) - \dots \quad (6.138)$$

$$\text{Or, } f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + O(h) \quad (6.139)$$

Approximate derivative by using the first term on the right-hand side of Eq. (6.139) is called the *first backward divided difference* because the two values of the function $f(x)$ at x_i and x_{i-1} are used to determine the derivative as shown in Fig. 6.18(a).

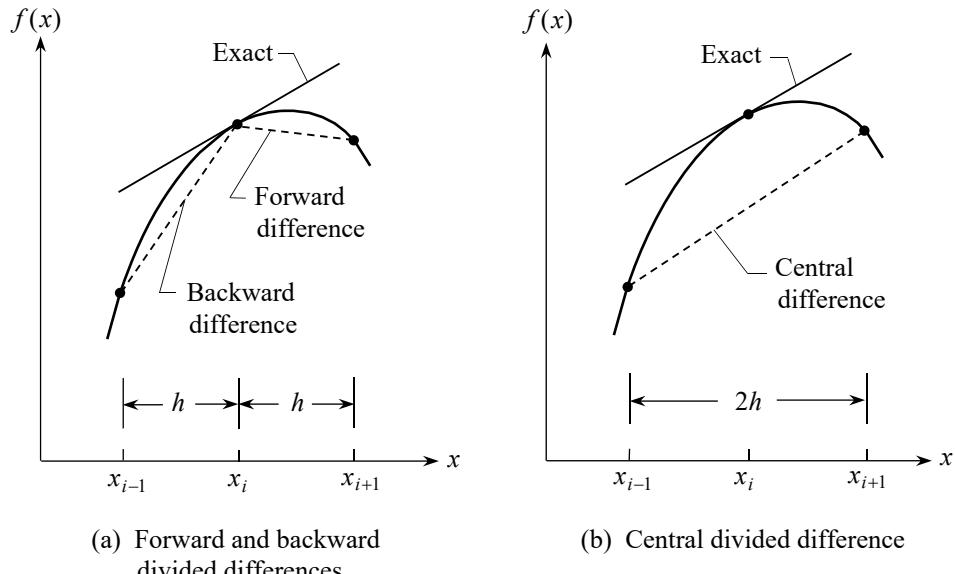


Figure 6.18 Comparisons between the approximate derivatives from the forward, backward and central divided-differences with the exact derivative.

By subtracting Eq. (6.137) from (6.134)

$$f(x_{i+1}) - f(x_{i-1}) = 2h f'(x_i) + \frac{2h^3}{3!} f'''(x_i) + \dots \quad (6.140)$$

the first derivative at x_i is

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} - \frac{h^2}{3!} f'''(x_i) + \dots \quad (6.141)$$

or,

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2h} + O(h^2) \quad (6.142)$$

The first derivative is estimated by using values of the function $f(x)$ at x_{i+1} and x_{i-1} . The approximate derivative by using the first term on the right-hand-side of Eq. (6.142) as shown in Fig. 6.18(b) is called the *central divided difference* which has the error of order h^2 .

Higher order derivatives of the function $f(x)$ can be derived in the same manner. For example, derivation of the second derivative may start from the Taylor's series for determining the function $f(x)$ at x_{i+2} from the values at x_i as follows.

$$f(x_{i+2}) = f(x_i) + (2h)f'(x_i) + \frac{(2h)^2}{2!} f''(x_i) + \dots \quad (6.143)$$

By multiplying the factor of two onto Eq. (6.134) and subtracting it from Eq. (6.143),

$$f(x_{i+2}) - 2f(x_{i+1}) = -f(x_i) + h^2 f''(x_i) + \dots \quad (6.144)$$

the second derivative of the function $f(x)$ at x_i is obtained as

$$f''(x_i) = \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{h^2} + O(h) \quad (6.145)$$

The first term on the right-hand side of Eq. (6.145) is called the *second forward divided difference* which can be used to approximate the second derivative of the function. The approximate value has the error of order h . Tables 6.4, 6.5 and 6.6 summarize the first- to fourth-order forward, backward and central divided differences of the function $f(x)$ evaluated at x_i , respectively.

Table 6.4 Forward divided differences with error of order $O(h)$.

$$\begin{aligned} f'(x_i) &= [f(x_{i+1}) - f(x_i)]/h \\ f''(x_i) &= [f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)]/h^2 \\ f'''(x_i) &= [f(x_{i+3}) - 3f(x_{i+2}) + 3f(x_{i+1}) - f(x_i)]/h^3 \\ f''''(x_i) &= [f(x_{i+4}) - 4f(x_{i+3}) + 6f(x_{i+2}) - 4f(x_{i+1}) + f(x_i)]/h^4 \end{aligned}$$

Table 6.5 Backward divided differences with error of order $O(h)$.

$$\begin{aligned} f'(x_i) &= [f(x_i) - f(x_{i-1})]/h \\ f''(x_i) &= [f(x_i) - 2f(x_{i-1}) + f(x_{i-2})]/h^2 \\ f'''(x_i) &= [f(x_i) - 3f(x_{i-1}) + 3f(x_{i-2}) - f(x_{i-3})]/h^3 \\ f''''(x_i) &= [f(x_i) - 4f(x_{i-1}) + 6f(x_{i-2}) - 4f(x_{i-3}) + f(x_{i-4})]/h^4 \end{aligned}$$

Table 6.6 Central divided differences with error of order $O(h^2)$.

$$\begin{aligned} f'(x_i) &= [f(x_{i+1}) - f(x_{i-1})]/2h \\ f''(x_i) &= [f(x_{i+1}) - 2f(x_i) + f(x_{i-1})]/h^2 \\ f'''(x_i) &= [f(x_{i+2}) - 2f(x_{i+1}) + 2f(x_{i-1}) - f(x_{i-2})]/2h^3 \\ f''''(x_i) &= [f(x_{i+2}) - 4f(x_{i+1}) + 6f(x_i) - 4f(x_{i-1}) + f(x_{i-2})]/h^4 \end{aligned}$$

The approximate derivatives presented in Tables 6.4 - 6.6 have errors of order h . For example, the first-order derivative of the function $f(x)$ as shown in Eq. (6.136) and in Table 6.4 has the error of order h . A more accurate derivative of the function $f(x)$ can be derived by starting from Eq. (6.135)

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2} f''(x_i) + O(h^2) \quad (6.146)$$

and substituting the second-order derivative from Eq. (6.145) into it to yield

$$f'(x_i) = \frac{-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)}{2h} + O(h^2) \quad (6.147)$$

The first-order derivative of the function $f(x)$ as obtained in Eq. (6.147) has the error of order h^2 . The same process as explained above can be applied to derive the high-order derivatives of the function $f(x)$. Their results by using the forward, backward and central divided differences are presented in Tables 6.7, 6.8 and 6.9, respectively.

Table 6.7 Forward divided differences with error of order $O(h^2)$.

$$\begin{aligned} f'(x_i) &= [-f(x_{i+2}) + 4f(x_{i+1}) - 3f(x_i)]/2h \\ f''(x_i) &= [-f(x_{i+3}) + 4f(x_{i+2}) - 5f(x_{i+1}) + 2f(x_i)]/h^2 \\ f'''(x_i) &= [-3f(x_{i+4}) + 14f(x_{i+3}) - 24f(x_{i+2}) + 18f(x_{i+1}) - 5f(x_i)]/2h^3 \\ f''''(x_i) &= [-2f(x_{i+5}) + 11f(x_{i+4}) - 24f(x_{i+3}) + 26f(x_{i+2}) - 14f(x_{i+1}) + 3f(x_i)]/h^4 \end{aligned}$$

Table 6.8 Backward divided differences with error of order $O(h^2)$.

$$\begin{aligned} f'(x_i) &= [3f(x_i) - 4f(x_{i-1}) + f(x_{i-2})]/2h \\ f''(x_i) &= [2f(x_i) - 5f(x_{i-1}) + 4f(x_{i-2}) - f(x_{i-3})]/h^2 \\ f'''(x_i) &= [5f(x_i) - 18f(x_{i-1}) + 24f(x_{i-2}) - 14f(x_{i-3}) + 3f(x_{i-4})]/2h^3 \\ f''''(x_i) &= [3f(x_i) - 14f(x_{i-1}) + 26f(x_{i-2}) - 24f(x_{i-3}) + 11f(x_{i-4}) - 2f(x_{i-5})]/h^4 \end{aligned}$$

Table 6.9 Central divided differences with error of order $O(h^4)$.

$$\begin{aligned} f'(x_i) &= [-f(x_{i+2}) + 8f(x_{i+1}) - 8f(x_{i-1}) + f(x_{i-2})]/12h \\ f''(x_i) &= [-f(x_{i+2}) + 16f(x_{i+1}) - 30f(x_i) + 16f(x_{i-1}) - f(x_{i-2})]/12h^2 \\ f'''(x_i) &= [-f(x_{i+3}) + 8f(x_{i+2}) - 13f(x_{i+1}) + 13f(x_{i-1}) - 8f(x_{i-2}) + f(x_{i-3})]/8h^3 \\ f''''(x_i) &= [-f(x_{i+3}) + 12f(x_{i+2}) - 39f(x_{i+1}) + 56f(x_i) - 39f(x_{i-1}) \\ &\quad + 12f(x_{i-2}) - f(x_{i-3})]/6h^4 \end{aligned}$$

Example 6.17 Determine the first-order derivative of the function in Fig. 6.3

$$f(x) = 2x^3 - 5x^2 + 3x + 1 \quad (6.148)$$

at $x = 1.0$ with $h = 0.1$ by using the forward divided difference $O(h)$, backward divided difference $O(h)$ and central divided difference $O(h^2)$. Compare the approximate derivatives with the exact solution.

It is noted that the exact derivative of the function $f(x)$ in Eq. (6.148) is

$$f'(x) = 6x^2 - 10x + 3 \quad (6.149)$$

Thus, the exact derivative value at $x = 1.0$ is

$$f'(x=1.0) = 6(1.0)^2 - 10(1.0) + 3 = -1.0 \quad (6.150)$$

By using $h = 0.1$, then

$$\begin{aligned} x_{i-1} &= 0.9 & ; & f(x_{i-1}) = 1.108 \\ x_i &= 1.0 & ; & f(x_i) = 1.000 \\ x_{i+1} &= 1.1 & ; & f(x_{i+1}) = 0.912 \end{aligned}$$

The above values are used to determine the approximate derivatives and their true errors according to those shown in Tables 6.4 to 6.6 as follows.

Derivative value and its error using the forward divided difference,

$$f'(1.0) = (0.912 - 1.000)/0.1 = -0.88 \quad ; \quad \varepsilon_t = 12\% \quad (6.151)$$

Derivative value and its error using the backward divided difference,

$$f'(1.0) = (1.000 - 1.108)/0.1 = -1.08 \quad ; \quad \varepsilon_t = 8\% \quad (6.152)$$

Derivative value and its error using the central divided difference,

$$f'(1.0) = (0.912 - 1.108)/0.2 = -0.98 \quad ; \quad \varepsilon_t = 2\% \quad (6.153)$$

The true errors of the approximate derivatives obtained from the forward, backward and central divided differences in Eqs. (6.151) - (6.153) show that the central divided difference provides higher solution accuracy. Such result agrees with the fact that the central divided difference gives the error of order h^2 , while both the forward and backward divided differences yield the error of order h .

Figure 6.19 presents a computer program for determining derivatives of a function at various x -locations by using the central divided difference. The program requires the input data of the beginning and ending locations including the number of locations for determining the derivatives. Typical results obtained from the program by using the function given in Example 6.17 are shown in Table 6.10.

```
% Program NumDif
% A numerical differentiation program.
func = @(x) ...
    (2.*x^3 - 5.*x^2 + 3.*x + 1.);
% Read end locations and no. of points:
a = input('Enter end location (a): ');
b = input('Enter end location (b): ');
n = input('Enter number of point: ');
h = (b - a)/(n - 1); x = a;
% Compute function values at points:
for i = 1:n
    fx(i) = func(x); x = x + h;
end
% Compute derivatives at points:
for i = 2:n-1
    diff(i) = (fx(i+1) - fx(i-1))/(2.*h);
end
diff(n) = (fx(n) - fx(n-1))/h;
fprintf('\n   x       f(x) ')
fprintf('      Derivative')
x = a;
for i = 1:n
    fprintf('\n %6.2f %12.3f %13.3f', ...
            x, fx(i), diff(i))
    x = x + h;
end
```

Figure 6.19 Computer program to determine derivatives of the function in Example 6.17 at various x -locations by using the central divided difference.

Table 6.10 Computed derivatives of the function in Example 6.17 at various x -locations by using the central divided difference.

x	$f(x)$	Derivative
0.50	1.500	-0.680
0.60	1.432	-0.820
0.70	1.336	-1.040
0.80	1.224	-1.140
0.90	1.108	-1.120
1.00	1.000	-0.980
1.10	0.912	-0.720
1.20	0.856	-0.340
1.30	0.844	0.160
1.40	0.888	0.780
1.50	1.000	1.120

6.12 MATLAB Commands for Differentiation

MATLAB has many useful commands to perform differentiation of a given function. One of the commands is `gradient` which is used to determine the gradient of the function $f(x, y)$ in the form of

$$\nabla f = \frac{\partial f}{\partial x} \hat{i} + \frac{\partial f}{\partial y} \hat{j}$$

If the given function f varies with only one independent variable x , then, the command determines the value of $\frac{df}{dx}$.

The format for using this command when the function f varies with one independent variable is

```
FX = gradient(function,h)
```

where FX is the computed gradients
 $function$ is the given function
 h is the step size

Example 6.18 Determine the derivatives of the function given in Example 6.17 at different x by using the MATLAB command `gradient`.

$$f(x) = 2x^3 - 5x^2 + 3x + 1 \quad (6.148)$$

The derivatives can be obtained by typing

```
>> f = inline('2*x.^3-5*x.^2+3*x+1', 'x');
>> x = 0.5:0.1:1.5;
>> y = f(x);
>> FX = gradient(y,0.1)
FX =
Columns 1 through 7
-0.6800 -0.8200 -1.0400 -1.1400 -1.1200 -0.9800 -0.7200
Columns 8 through 11
-0.3400 0.1600 0.7800 1.1200
```

The computed derivatives are identical to those presented in Table 6.10.

6.13 Closure

Numerical methods for integration and differentiation of a given function are presented in this chapter. Fundamentals of these numerical methods are explained in details with examples and computer programs. The simplest integration method is based on the trapezoidal rule. The trapezoidal rule approximates the integral from the area under a linear function. A more accurate integral solution can be obtained by using the composite trapezoidal rule. The composite trapezoidal rule divides the integration range into a number of segments so that the single trapezoidal rule is applied to each segment. If the integral is approximate by the area under a quadratic function, the method is called the Simpson's rule. Improved integral solution accuracy can be obtained by determining the area under higher order polynomials. These methods lead to a family of the methods so called the Newton-Cotes formulas. Other integration methods that can provide high solution accuracy, such as the Romberg and Gauss integration methods, are also presented. The Gauss integration method is popular and widely used in many commercial software for solving practical problems. Numerical integration methods for two- and three-dimensional domains are also presented and explained. These methods follow the procedure used for one-dimensional domain by performing the integration in each coordinate direction, one at a time.

Numerical methods for determining derivatives of a given function at various locations are presented at the end of the chapter. The approximate derivatives can be determined by using the forward, backward and central divided differences. Understanding the methods for integration and differentiation is essential to further study the higher-level numerical methods in the following chapters.

Exercises

- Determine the integral

$$I = \int_2^8 (4x^5 - 3x^4 + x^3 - 6x + 2) dx$$

by using the trapezoidal rule and composite trapezoidal rule with $n = 2, 4$ and 6 . Plot distribution of the integrand between the given lower and upper limits. Compare the computed values with the exact solution and determine the true percentage errors.

- Determine the integral

$$I = \int_0^{0.5} \frac{dx}{1+x^4}$$

by using the trapezoidal rule and composite trapezoidal rule with $n = 2$ and 5 . Compare the computed values with the exact solution and determine their true percentage errors. Check the computed values with the solutions obtained by modifying the computer program in Fig. 6.6.

3. Determine the integral

$$I = \int_0^4 x e^{2x} dx$$

by using the composite trapezoidal rule with $n = 2, 4$ and 8 . Compare the computed values with the exact solution and determine the true percentage errors.

4. Determine the integral

$$I = \int_{-\pi}^{\pi} \frac{dx}{1 + \sin^2 x}$$

by using the trapezoidal rule and composite trapezoidal rule with $n = 2, 4$ and 6 . Compare the computed values with the exact solution and determine the true percentage errors.

5. Determine the integral

$$I = \int_1^5 \frac{x e^x}{(x+1)^2} dx$$

by using the trapezoidal rule and composite trapezoidal rule with $n = 2, 4$ and 5 . Compare the computed values with the solutions obtained by modifying the computer program in Fig. 6.6.

6. Determine the elliptic integral of the first kind as shown in Eq. (6.5)

$$K(\theta) = \int_0^{\pi/2} \frac{dx}{\sqrt{1 - \sin^2 \frac{\theta}{2} \sin^2 x}}$$

when $\theta = \pi/6$ by using the trapezoidal rule and composite trapezoidal rule with $n = 2, 3$ and 6 . Compare the computed values with the solutions obtained by modifying the computer program in Fig. 6.6. Determine the number of segments n required to provide the solution accuracy up to 4 significant figures.

7. Derive the integral expression by using the Simpson's 1/3 rule as shown in Eq. (6.43). The derivation starts by integrating the second-order Lagrange polynomial in Eq. (6.42). Show the derivation in details.
8. Show that the solution error produced by using the Simpson's 1/3 rule for integrating a function f from the limit a to b is

$$E_t = -\frac{(b-a)^5}{2,880} f^{(4)}(\xi)$$

Then, explain physical meaning of the error expression above.

9. Determine the integral

$$I = \int_{-1}^2 (x^7 + 2x^3 - 1) dx$$

by using the Simpson's 1/3 rule when $n = 2, 4$ and 6 . Compare the computed values with the exact solution and determine the true percentage errors. Then, solve the problem again but by using by using the Simpson's 3/8 rule with $n = 3, 6$.

10. Determine the integral

$$I = \int_0^{\pi/4} \frac{x}{\cos^2 x} dx$$

by using the Simpson's 1/3 rule when $n = 2, 4$. Show the computational procedure in details. Solve the problem again but by using the Simpson's 3/8 rule with $n = 3$ and 6 .

11. Determine the integral

$$I = \int_0^{\pi/2} \frac{\sin x}{\sqrt{1 - 0.25 \sin^2 x}} dx$$

by using the Simpson's 1/3 rule when $n = 2, 4$ and 6 . Show the computational procedure in details. Check the computed values with those obtained by modifying the computer program in Fig. 6.9.

12. Develop a computer program to integrate a function by using the Simpson's 3/8 rule. Verify the program by solving Example 6.8. Then, use the program to determine the integral

$$I = \int_0^1 \frac{e^x}{1+e^x} dx$$

with $n = 3, 30$ and 300 , respectively. Provide comments on the accuracy of the computed values.

13. Show that the solution error that occurs by using the composite Simpson's 1/3 rule with n segments to integrate a function f from the lower limit a to upper limit b is

$$E_a = -\frac{(b-a)^5}{180n^4} \bar{f}^{(4)}$$

where $\bar{f}^{(4)}$ is the average fourth-order derivative of all segments. Then, set up an example to demonstrate the validity of the error expression above.

14. Derive Eqs. (6.56) and (6.60) for determining the integral of a given function and the associated error, respectively, using the Simpson's 3/8 rule. Show detailed derivation and explain physical meaning of the terms in these two equations.

15. In practice, the integral values are determined from a set of experimental data because their continuous functions may not be available. Modify the computer programs of the composite trapezoidal and Simpson's rules as shown in Figs. 6.6 and 6.9 to determine the integral values of the data below from the limits $a = 0$ to $b = 2$.

x	0	0.2	0.4	0.6	0.8	1.0
$f(x)$	-1.000	-0.992	-0.936	-0.784	-0.488	0.000
x	1.0	1.2	1.4	1.6	1.8	2.0
$f(x)$	0.000	0.728	1.744	3.096	4.832	7.000

The modified computer programs should be validated by solving the integral below that has exact solution.

$$I = \int_0^2 f(x) dx = \int_0^2 (x^3 - 1) dx$$

16. Derive the integral expression and associated error that occur by using the Boole's rule as shown in Eq. (6.68). Show the derivation in details and set up an example such that the Boole's rule can not provide exact integral solution.
17. Prove the Romberg integration formula in Eq. (6.87) by showing the derivation in details. Explain physical meanings of the integral expressions when $k = 1, 2, 3$ and 4 , respectively.
18. Study the computer program for the Romberg integration in Fig. 6.11. Draw a flow chart and explain the computational procedure that occurs in the program. Then, use the program to solve Example 6.10 and discuss on the accuracy of the computed integral value.
19. Apply the Romberg integration method to determine the integral in Eq. (6.4)

$$I = \int_0^{0.8} e^{-x^2} dx$$

Show the solution procedure so that the computed integral value has the relative error less than 0.0001%.

20. Apply the Romberg integration method for 4 rounds to determine the integral

$$I = \int_{\pi/4}^{\pi} (e^{2x} + \cos x) dx$$

Compare the solution obtained from each round with the exact solution.

21. Apply the Romberg integration method for 3 rounds to determine the integral

$$I = \int_0^{\pi/4} e^{3x} \sin 2x \, dx$$

Show the solution procedure in details. Compare the computed integral values with those obtained by using the computer program in Fig. 6.11.

22. Apply the Romberg integration method for 4 rounds to determine the integral

$$I = \int_1^{1/2} e^{-x} \, dx$$

Show the computational procedure in details.

23. Determine the elliptic integral of the first kind in problem 6 again but by using the Romberg integration method. Apply the method until computed solution has the relative error less than 0.0001%. Verify the computed solution with that obtained from modifying the computer program in Fig. 6.11.

24. Derive the weights and locations of the 3-point Gauss integration method as shown in Eq. (6.105a-c). Explain necessary conditions required to derive these values.

25. Determine the integral in Example 6.11 again but by using the 3-point Gauss integration method. Show the solution procedure in details. Is it possible that the solution accuracy decreases with the increased number of Gauss point? If the answer is yes, explain the source of the error.

26. Modify the computer program in Fig. 6.15 for determining the integral

$$I = \int_0^{\pi/2} \sin x \, dx$$

It is noted that the above integral was determined by using the Romberg integration method in Example 6.10. Provide comments on the solution accuracy obtained from using the two methods.

27. Apply the Gauss integration method to determine

$$I = \int_3^4 \frac{x}{\sqrt{x^2 + 4}} \, dx$$

by using the number of Gauss points of $n = 2, 3$ and 4 . Show the computational procedure in details.

28. Apply the Gauss integration method to determine

$$I = \int_0^1 x^2 e^{-x} dx$$

by using the number of Gauss points of $n = 2, 3$ and 4 . Show the computational procedure in details. Compare the computed integral values with those obtained by modifying the computer program in Fig. 6.15.

29. Apply the Gauss integration method to determine

$$I = \int_0^1 (1+x^2)^{3/2} dx$$

by using the number of Gauss points of $n = 2, 3, 4, 5$ and 6 . Compare the computed integral values with the exact solution of 1.567951962 .

30. Apply the Gauss integration method to determine

$$I = \int_0^5 \frac{dx}{1+(x+\pi)^2}$$

by using the number of Gauss points of $n = 2, 3, 4, 5$ and 6 . Compare the computed integral values with the exact solution of

$$I = \tan^{-1}(5+\pi) - \tan^{-1}(\pi)$$

Then, give comment on the possibility that the Gauss integration method can provide exact integral solution. If it is possible, how many Gauss points are needed?

31. The computer program for the Gauss integration method was developed so that it is easy to understand. The program does not take the advantage that the weights W_i and locations ξ_i are symmetric. The data statements for the weights and locations can be reduced by using their symmetrical property. Improve the program by using the symmetrical property of these data in order to reduce the memory requirement. Then validate the program by solving the integrals in Examples 6.10 and 6.11.

32. Use the `trapz` command in MATLAB to determine the integral

$$I = \int_0^2 \sqrt{2x-x^2} dx$$

Select appropriate segment length so that the true error is less than 0.0001 . Check the accuracy of the computed integral value by comparing with the exact solution determined from

$$\int \sqrt{2x-x^2} dx = \frac{x-1}{2} \sqrt{2x-x^2} + \frac{1}{2} \arcsin(x-1)$$

33. Use the `trapz` command in MATLAB to determine the integral

$$I = \int_1^3 \frac{1}{1+2\sin x} dx$$

Select appropriate segment length so that the true error is less than 0.0001. Derive the exact integral solution to measure the accuracy of the computed integral value.

$$\int \frac{1}{1+2\sin x} dx = \frac{1}{\sqrt{3}} \ln \left[\frac{\tan(x/2) + 2 - \sqrt{3}}{\tan(x/2) + 2 + \sqrt{3}} \right]$$

34. Use the `trapz` command in MATLAB to determine the integral

$$I = \int_0^3 \frac{1}{\sqrt{2x^2 + 3x + 1}} dx$$

Select appropriate segment length so that the true error is less than 0.0001. Check the accuracy of the computed integral value by comparing with the exact solution determined from

$$\int \frac{1}{\sqrt{2x^2 + 3x + 1}} dx = \frac{1}{\sqrt{2}} \ln \left| 2\sqrt{4x^2 + 6x + 2} + 4x + 3 \right|$$

35. Use the single trapezoidal rule to determine the integral

$$I = \int_0^3 \int_1^4 (x^2 y + 8y) \sin x dx dy$$

Repeat the problem but by using the Simpson's 1/3 rule. Compare the accuracy of the computed integral values with the exact solution.

36. Modify the computer program that uses the composite trapezoidal rule to determine the double integral

$$I = \int_0^\pi \int_1^3 \frac{\cos x (2y^2 + \sin x)}{\sqrt{1+y}} dx dy$$

Use the number of segments of 5, 10 and 20 in each x - and y -direction to verify the convergence of the computed integral values.

37. Apply the Gauss integration to determine the double integral

$$I = \int_2^3 \int_1^2 e^x e^y dx dy$$

Use $n=2$ in each direction. Repeat the problem but by using $n=3$ to verify the convergence of the computed integral values.

38. Determine the integral in Problem 36 again but by using the `dblquad` command in MATLAB. Use the required tolerance of 1×10^{-5} and compare the solution with that obtained from Problem 36.

39. Use the `dblquad` command in MATLAB to determine the double integral

$$I = \int_{-2}^3 \int_2^4 2x^2y + 3xy^2 \, dx \, dy$$

Specify an appropriate tolerance and compare the computed integral value with the exact solution of $910/3$.

40. Use the `dblquad` command in MATLAB to determine the double integral

$$I = \int_0^{0.5\pi} \int_0^{\pi} x \cos(xy) \cos^2(\pi x) \, dy \, dx$$

Specify an appropriate tolerance and compare the computed integral value with the exact solution of $1/(3\pi)$.

41. Use the single trapezoidal rule to determine the triple integral

$$I = \int_2^3 \int_0^1 \int_1^2 (1+x)yz^2 \, dx \, dy \, dz$$

Then, repeat the problem but by using the Gauss integration method with $n = 2$ in each x -, y - and z -direction. Compare the computed integral value with the exact solution of $95/12$.

42. Use the `triplequad` command in MATLAB to determine the triple integral

$$I = \int_0^{\pi/2} \int_0^1 \int_0^2 zr^2 \sin \theta \, dz \, dr \, d\theta$$

Specify an appropriate tolerance and compare the computed integral value with the exact solution of $2/3$.

43. Use the `triplequad` command in MATLAB to determine the triple integral

$$I = \int_0^{2\pi} \int_0^\pi \int_0^5 (\rho^4 \sin \phi) \, d\rho \, d\phi \, d\theta$$

Specify an appropriate tolerance and compare the computed integral value with the exact solution of $2,500\pi$.

44. Show the derivation for the expressions of the first and second derivatives as given in Tables 6.4 - 6.6 in details.

45. Show the derivation for the expression of the third derivative in Table 6.6 by using the central divided difference.

46. Determine the first derivative of the function

$$f(x) = e^x$$

at $x=2$ with $h=0.25$ by using the forward divided difference $O(h)$, backward divided difference $O(h)$ and central divided difference $O(h^2)$. Show detailed derivation and determine the true errors of the computed derivatives with the exact solution.

47. Determine the first and second derivatives of the function

$$f(x) = \tan^{-1}(x^2 - x + 1)$$

at $x=1$ with $h=0.1$ by using the forward divided difference $O(h)$, backward divided difference $O(h)$ and central divided difference $O(h^2)$. Then, repeat the problem again but by using $h=0.05$.

48. Show the derivation for the expressions of the first and second derivatives as given in Tables 6.7 - 6.9 in details.

49. Determine the first to the fourth derivatives of the function

$$f(x) = e^{x/3} + x^2$$

at $x=-2.5$ with $h=0.1$ by using the forward divided difference $O(h^2)$, backward divided difference $O(h^2)$ and central divided-difference $O(h^4)$. Show detailed derivation and determine the true errors of the computed derivatives with the exact solution.

50. Modify the computer program for determining the first derivative as shown in Fig. 6.19 by using the more accurate derivative expressions in Tables 6.7 - 6.9. Then, use the program to solve Example 6.17 and compare the computed solutions with those shown in the example.

Chapter

7

Ordinary Differential Equations

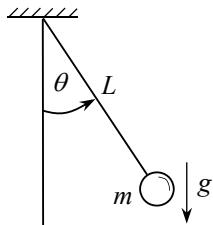
7.1 Introduction

Ordinary differential equations occur in solving many scientific and engineering problems. For example, in the determination of the shuttle velocity v that varies with time t during descending through the earth atmosphere, the governing differential Eq. (1.5) representing the Newton's second law is

$$\frac{dv}{dt} = g - \frac{c}{m}v \quad (7.1)$$

where g is the gravitational acceleration constant, c is the air drag coefficient and m is the mass of the shuttle. Equation (7.1) is called an ordinary differential equation (ODE) because the dependent variable v varies only with the independent variable t . The ordinary differential equation differs from the partial differential equation (PDE) which is presented in the following chapter. A dependent variable in the partial differential equation varies with two or more independent variables. Solutions to the partial differential equations thus are more complex and difficult to derive.

The ordinary differential equation can be classified into several types. Equation (7.1) is called the first-order ordinary differential equation because the highest derivative term is of the first-order. Some ordinary differential equations contain the second-order derivative terms. For example, the equilibrium equation of a swinging pendulum as shown in Fig. 7.1 is in the form of the second-order ordinary differential equation. The equation is derived from the equilibrium condition at any instant during swinging by using the Newton's second law.



The equilibrium condition of the forces acting on the mass in the tangential direction leads to the second-order differential equation in the form,

$$\frac{d^2\theta}{dt^2} + \frac{g}{L} \sin \theta = 0 \quad (7.2)$$

where θ is the swinging angle that varies with time t and L is the cord length.

Figure 7.1 Swinging pendulum.

It is noted that the second-order differential equation as shown in Eq. (7.2) can be separated into two first-order differential equations by introducing a new variable. For example, if

$$\frac{d\theta}{dt} = \beta \quad (7.3a)$$

then, Eq. (7.2) becomes

$$\frac{d\beta}{dt} = -\frac{g}{L} \sin \theta \quad (7.3b)$$

Because a second-order differential equation can be separated into two first-order differential equations, the numerical methods for solving the first-order differential equation are presented in this chapter. The same methods can be applied to solve the second-order and other higher-order ordinary differential equations.

Equations (7.1) and (7.2) as shown above are linear and nonlinear ordinary differential equations, respectively. A linear differential equation can be identified by considering the differential equation in the general form of

$$a \frac{d^2y}{dx^2} + b \frac{dy}{dx} + cy = d \quad (7.4)$$

If the coefficients a, b, c, d are constant or function of the independent variable x , the equation is *linear*. Equation (7.4) becomes *nonlinear* if the coefficients are function of the dependent variable y , such as $a = y^2$ or $b = 2y$. Knowing that a differential equation is either linear or nonlinear is useful for finding its solution. Exact solutions for most nonlinear differential equations cannot be derived. Numerical methods are the only way to find their approximate solutions. Since most of the differential equations that occur in practical problems are nonlinear, the numerical methods thus help analysts to obtain solutions that could not be found in the past.

Equation (7.2) that governs the swinging phenomenon of the pendulum is a nonlinear differential equation. The equation is nonlinear because the $\sin \theta$ function can be expressed in form of infinite series of the dependent variable θ as

$$\sin \theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \dots \quad (7.5)$$

Thus, the exact solution to the differential Eq. (7.2) is difficult or even impossible to derive. Exact solution to Eq. (7.2) can be obtained easier if the differential equation becomes linear. The differential equation becomes linear if

$$\sin \theta = \theta \quad (7.6)$$

which is valid for small swinging angle θ . By substituting Eq. (7.6) into Eq. (7.2), the linear differential equation is

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\theta = 0 \quad (7.7)$$

Its exact solution for the swinging angle θ that varies with time t is

$$\theta(t) = \theta_0 \cos \sqrt{\frac{g}{L}} t \quad (7.8)$$

The exact solution above is obtained by using the initial angle θ_0 and zero velocity at time $t = 0$ as

$$\theta(t=0) = \theta_0 \quad \text{and} \quad \frac{d\theta}{dt}(t=0) = 0 \quad (7.9)$$

The example shows that two initial conditions are required to solve a second-order differential equation. Similarly, an initial condition is needed for solving a first-order differential equation such as Eq. (7.1). Solutions to the ordinary differential equation thus depend on the initial conditions of the problems as shown by the following example.

A first-order ordinary differential equation can be written in a general form as,

$$\frac{dy}{dx} = f(x, y) \quad (7.10)$$

As an example, if

$$f(x, y) = 3x^2 - 2x + 1 \quad (7.10a)$$

then,

$$\frac{dy}{dx} = 3x^2 - 2x + 1 \quad (7.10b)$$

The general solution is obtained after performing integration,

$$y = x^3 - x^2 + x + C \quad (7.10c)$$

where C is the integrating constant. The integration constant C is determined from the initial condition of the problem. As an example, if the initial condition is given by $y(x=0)=2$, then $C=2$. Thus, the solution to the differential equation with the given initial condition is

$$y = x^3 - x^2 + x + 2 \quad (7.10d)$$

Equation (7.10d) shows that the final solution of a differential equation depends on the initial condition. The same solution behavior occurs for the swinging pendulum problem where its final solution depends on the initial angle θ_0 .

In this chapter, popular methods for solving the ordinary differential equations are presented. These methods are: (1) the Euler's method, (2) the Heun's method, (3) the modified Euler's method, (4) the Runge-Kutta method, (5) the methods for solving a system of first-order differential equation, and (6) the multistep methods. Corresponding computer programs are also presented to aid understanding of these methods for obtaining solutions.

7.2 Euler's Method

The Euler's method is the simplest method for solving the ordinary differential equation in the form,

$$\frac{dy}{dx} = f(x, y) \quad (7.10)$$

The concept of the method is explained by considering Fig. 7.2.

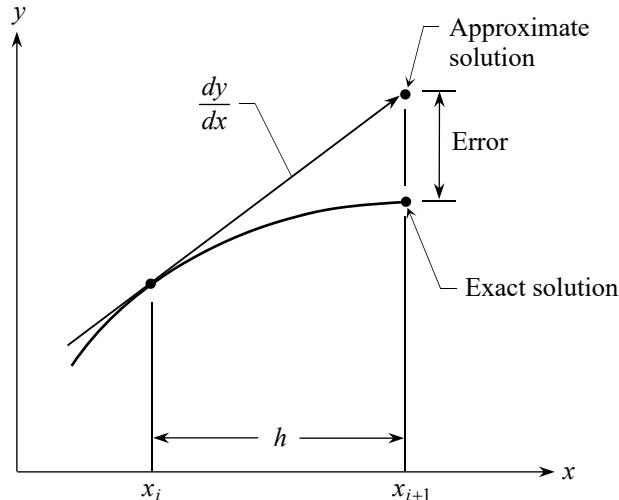


Figure 7.2 The Euler's method.

From Fig. 7.2, the approximate solution y_{i+1} at x_{i+1} is determined from the solution y_i at x_i by estimating the slope from

$$\frac{dy}{dx} = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = \frac{y_{i+1} - y_i}{h} \quad (7.11)$$

where $h = x_{i+1} - x_i$ is the step size used in the computation. By substituting the slope from Eq. (7.11) into Eq. (7.10),

$$\begin{aligned} \frac{y_{i+1} - y_i}{h} &= f(x_i, y_i) \\ \text{i.e.,} \quad y_{i+1} &= y_i + f(x_i, y_i)h \end{aligned} \quad (7.12)$$

Equation (7.12) shows that the approximate solution y_{i+1} is determined from the solution y_i at x_i by using the step size h . As shown in the figure, the solution error varies directly with the step size h . Higher solution accuracy is obtained by using smaller step size as demonstrated in the following example.

Example 7.1 Use the Euler's method to solve the first-order differential equation,

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0) = 1$. Determine the approximate solutions by using the three different step sizes of $h = 1.00, 0.50$ and 0.25 . Compare the Euler's solutions with the exact solution.

The exact solution to the differential Eq. (7.13) can be derived by separating the two variables onto opposite sides of the equation and performing integration,

$$\int \frac{dy}{y} = \int \cos x \, dx \quad (7.14)$$

The integration yields,

$$\ln y = \sin x + A \quad (7.15)$$

where A is the integrating constant which can be determined from the initial condition of $y(0) = 1$,

$$\ln 1 = 0 + A$$

to give,

$$A = \ln 1 = 0$$

Then, Eq. (7.15) becomes

$$\ln y = \sin x$$

leading to the exact solution of

$$y = y(x) = e^{\sin x} \quad (7.16)$$

To determine the Euler's solution, Eq. (7.12) is employed together with the given function $f(x, y)$,

$$y_{i+1} = y_i + (y_i \cos x_i) h \quad (7.17)$$

By starting from the initial condition of $y(x=0) = 1$, i.e., $x_0 = 0$ and $y_0 = 1$, then Eq. (7.17) gives

$$y_1 = 1 + [(1)(1)](1) = 2 \quad (7.18a)$$

so that the exact error is

$$E_t = e^{\sin(1)} - 2 = 0.31978 \quad (7.18b)$$

The approximate solution obtained from Eq. (7.18a) is used to determine the solution for the second round. The Euler's Eq. (7.17) is used again with $x_1 = 1$ and $y_1 = 2$,

$$y_2 = 2 + [(2)(0.54030)](1) = 3.08060 \quad (7.19a)$$

for which the exact error is

$$E_t = e^{\sin(2)} - 3.08060 = -0.59802 \quad (7.19b)$$

The process is repeated to determine the approximate solution at the third round with $x_2 = 2$ and $y_2 = 3.08060$ to give

$$y_3 = 3.08060 + [(3.08060)(-0.41615)](1) = 1.79862 \quad (7.20a)$$

and the exact error is,

$$E_t = e^{\sin(3)} - 1.79862 = -0.64706 \quad (7.20b)$$

After three rounds of computation, the approximate solutions are obtained and compared with the exact solution as shown in Fig. 7.3.

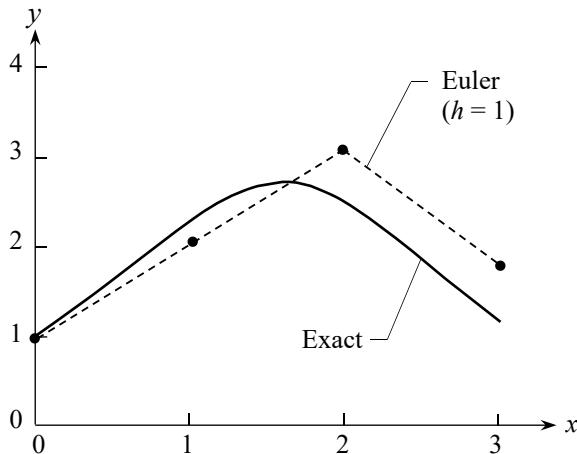


Figure 7.3 Comparison between the Euler's solution using $h = 1$ with the exact solution in Example 7.1.

Accuracy of the approximate solutions obtained from the Euler's method increases by reducing the step size h . Table 7.1 shows the approximate solutions obtained from using the three different step sizes of $h = 1.00, 0.50$ and 0.25 as compared to the exact solution. These approximate solutions are compared with the exact solution as plotted in Fig. 7.4.

Table 7.1 Comparison of the Euler's solutions by using different step sizes with the exact solution.

x	Exact solution	Euler's solutions		
		$h = 1.00$	$h = 0.50$	$h = 0.25$
0.00	1.00000	1.00000	1.00000	1.00000
0.25	1.28070			1.25000
0.50	1.61515		1.50000	1.55279
0.75	1.97712			1.89346
1.00	2.31978	2.00000	2.15819	2.23982
1.25	2.58309			2.54236
1.50	2.71148		2.74122	2.74278
1.75	2.67510			2.79128
2.00	2.48258	3.08060	2.83818	2.66690
2.25	2.17727			2.38944
2.50	1.81934		2.24763	2.01419
2.75	1.46472			1.61078
3.00	1.15156	1.79862	1.34729	1.23857

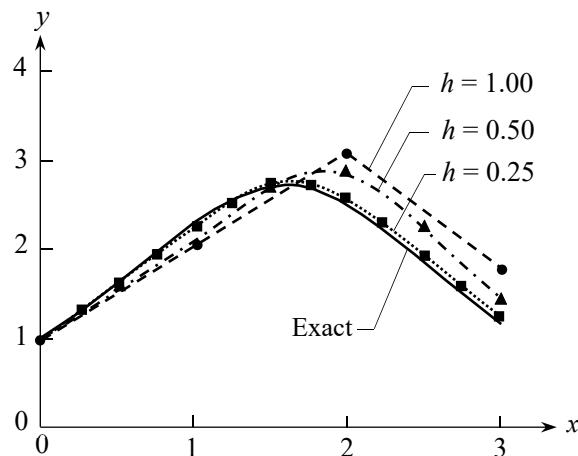


Figure 7.4 Comparison of the Euler's solutions using three different step sizes with the exact solution in Example 7.1.

Even though the Euler's method is simple, developing a computer program is necessary for obtaining solutions for a large number of steps. This is because the computational procedure is repeated by using the values obtained from the earlier steps. Miscalculating a value thus affects the solutions at the later steps. Figure 7.5 shows a computer program of the Euler's method for obtaining the solutions in Example 7.1.

```
% Program Euler
% A program for solving an ordinary
% differential equation by using the
% Euler's method.
% Provide the f(x,y) function:
func = @(x,y)(y*cos(x));
% Read the initial conditions, number of
% steps and step size:
x(1) = input( ...
    '\nEnter initial value of x: ');
y(1) = input( ...
    '\nEnter initial value of y: ');
n = input( 'Enter number of steps: ');
h = input( 'Enter the step size: ');
fprintf('\n      x          y');
fprintf('\n%16.6e',x(1),y(1));
for i = 1:n
    slope = func(x(i),y(i));
    y(i+1) = y(i) + slope*h;
    x(i+1) = x(i) + h;
    fprintf('\n%16.6e',x(i+1),y(i+1));
end
```

Figure 7.5 Computer program for solving the first-order ordinary differential equation by using the Euler's method in Example 7.1.

Solutions obtained from the Euler's method as shown in Table 7.1 and Fig. 7.4 indicate that their errors vary with the step size h . Such errors consist of: (1) the local error from using the Euler's formula in Eq. (7.12) and (2) the error that is accumulated as the Euler's process repeats. Combination of the two errors is known as the global error. Magnitude of the global error varies in the same order with the step size h , or $O(h)$. If the step size is reduced by a half, the global error also reduces by a half. The Euler's method is thus sometimes called the first-order method.

In the next section, another method that can provide solution with the second-order of accuracy, $O(h^2)$, is presented. The method reduces the solution error into a quarter if the step size h is cut by a half.

7.3 Heun's Method

The Heun's method is modified from the Euler's method to increase the solution accuracy. Figure 7.2 of the Euler's method suggests that the solution accuracy increases if a proper slope ($y' = dy/dx$) is employed in the computation. Since the computed slope at x_i from the Euler's method is

$$y'_i = f(x_i, y_i) \quad (7.21)$$

Such slope is used for determining the solution at x_{i+1} from

$$y_{i+1}^0 = y_i + f(x_i, y_i)h \quad (7.22)$$

The approximate solution obtained from Eq. (7.22) can be employed to determine the slope at the same location x_{i+1} as shown in Fig. 7.6(a) as

$$y'_{i+1} = f(x_{i+1}, y_{i+1}^0) \quad (7.23)$$

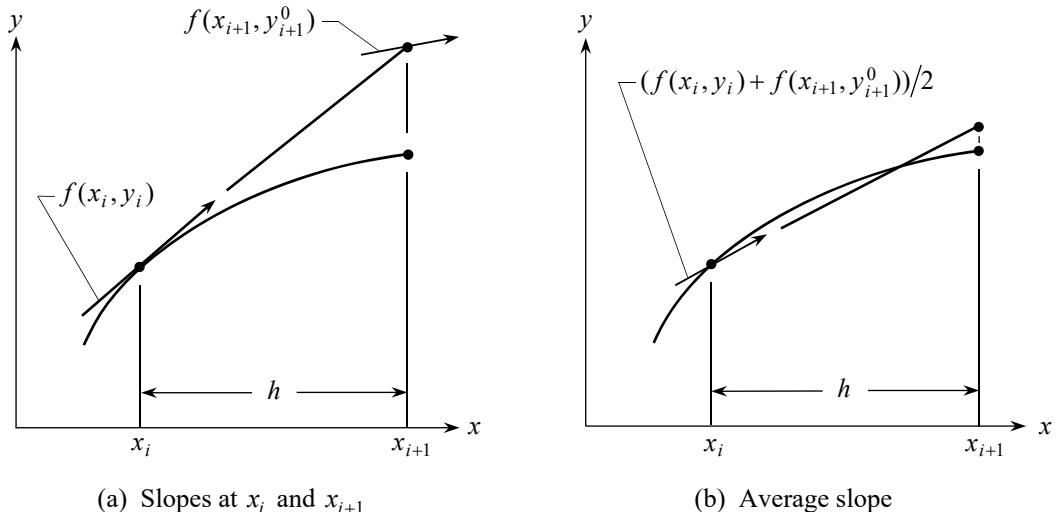


Figure 7.6 The Heun's method.

The new slope at x_{i+1} is then averaged with the slope at x_i . The average slope is then used at x_i for determining the solution at x_{i+1} as shown in Fig. 7.6(b). The process should provide improved solution at x_{i+1} because a more accurate slope is employed in the computation.

From Eqs. (7.21) and (7.23), the average slope is

$$\bar{y}' = \frac{y'_i + y'_{i+1}}{2} = \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} \quad (7.24)$$

Thus, the approximate solution at x_{i+1} is

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} h \quad (7.25)$$

In summary, the Heun's method consists of two computational steps. The first step is to determine the slope from the predicted solution y_{i+1}^0 at x_{i+1} . The process of this first step is known as the predictor. The predicted slope y_{i+1}^0 is averaged with the slope at x_i in the second step. The averaged slope is then used at x_i to determine the solution at x_{i+1} . The second step is called the corrector that can provide a more accurate solution as compared to the original Euler's method. The Heun's method is thus sometimes called the predictor-corrector method because of the two processes above are employed. These two processes are summarized as follows:

$$\text{Predictor: } y_{i+1}^0 = y_i + f(x_i, y_i) h \quad (7.26a)$$

$$\text{Corrector: } y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} h \quad (7.26b)$$

Example 7.2 Use the Heun's method to solve the same first-order ordinary differential equation

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0) = 1$. The problem was solved earlier by the Euler's method in Example 7.1 by employing the step size of $h = 1.00$. Compare the solution with the Euler's and exact solutions. Then, develop a computer program to obtain a more accurate solution by using a smaller step size of $h = 0.25$.

For the first round at $x_0 = 0$ and $y_0 = 1$ with $h = 1$, the predictor from Eq. (7.26a) is

$$y_1^0 = 1 + [(1)(1)](1) = 2 \quad (7.27a)$$

Then, the slope at the end of the step is

$$f(x_1, y_1^0) = (2) \cos(1) = 1.08060$$

Thus, the corrector which is the solution at the end of the step can be determined from Eq. (7.26b)

$$y_1 = 1 + \frac{1 + 1.08060}{2}(1) = 2.04030 \quad (7.27b)$$

so that the exact error is

$$E_t = e^{\sin(1)} - 2.04030 = 0.27948 \quad (7.27c)$$

For the second round $x_1 = 1$, $y_1 = 2.04030$, the predictor from Eq. (7.26a) is

$$y_2^0 = 2.04030 + [2.04030 \cos(1)](1) = 3.14268 \quad (7.28a)$$

The slope at the end of the step, i.e. at $x_2 = 2$, is

$$f(x_2, y_2^0) = (3.14268) \cos(2) = -1.30782$$

Then, with the corrector from Eq. (7.26b),

$$y_2 = 2.04030 + \frac{1.10238 - 1.30782}{2}(1) = 1.93758 \quad (7.28b)$$

which gives the exact error of

$$E_t = e^{\sin(2)} - 1.93758 = 0.54500 \quad (7.28c)$$

For the third round starting from $x_2 = 2$, $y_2 = 1.93758$, the predictor, the slope, the corrector and the exact error are,

$$y_3^0 = 1.93758 + [1.93758 \cos(2)](1) = 1.13126 \quad (7.29a)$$

$$f(x_3, y_3^0) = (1.13126) \cos(3) = -1.11994$$

$$y_3 = 1.93758 + \frac{-0.80632 - 1.11994}{2}(1) = 0.97445 \quad (7.29b)$$

$$E_t = e^{\sin(3)} - 0.97445 = 0.17711 \quad (7.29c)$$

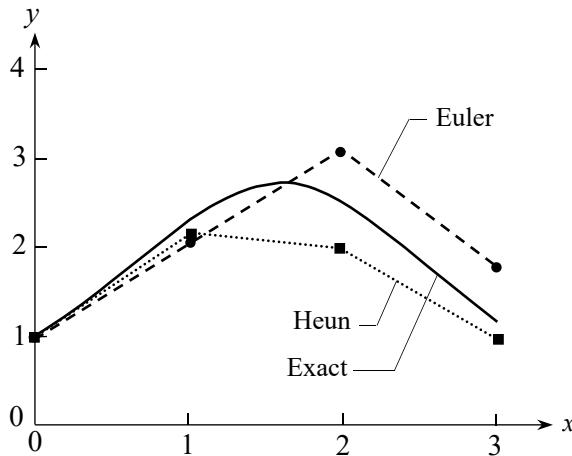


Figure 7.7 Comparison of the Heun's solution using $h = 1$ with the Euler's and exact solutions in Example 7.2.

The Heun's solution is plotted to compare with the Euler's and exact solutions as shown in Fig. 7.7. Figure 7.8 presents a computer program of the Heun's method for solving the differential equation in Example 7.2. The program is similar to that of the Euler's method in Fig. 7.5 except few additional commands are included for determining the predictor and corrector. Table 7.2 shows the Heun's solution which is more accurate than the Euler's solution. The Heun's method is the second-order method that provides second-order accuracy of solution, $O(h^2)$. It is noted that the Euler's method as explained earlier can be modified to yield the second-order solution accuracy similar to the Heun's method as explained in the following section.

```
% Program Heun
% A program for solving an ordinary
% differential equation by using the
% Heun's method.
% Provide the f(x,y) function:
func = @(x,y)(y*cos(x));
% Read the initial conditions, number of
% steps and step size:
x(1) = input('
    'nEnter initial value of x: ');
y(1) = input('
    'Enter initial value of y: ');
n = input('Enter number of steps: ');
h = input('Enter the step size: ');
fprintf('\n      x          y');
for i = 1:n
    s0 = func(x(i),y(i));
    y1 = y(i) + s0*h;
    x(i+1) = x(i) + h;
    s1 = func(x(i+1),y1);
    sa = (s0 + s1)/2.;
    y(i+1) = y(i) + sa*h;
    fprintf('\n%16.6e%16.6e',x(i+1),y(i+1));
end
plot(x,y,'-o'), hold on
% Exact solution:
xe = 0:.05:3; ye = exp(sin(xe));
plot(xe,ye,'-b')
```

Figure 7.8 Computer program for solving the first-order ordinary differential equation by using the Heun's method in Example 7.2.

7.4 Modified Euler's Method

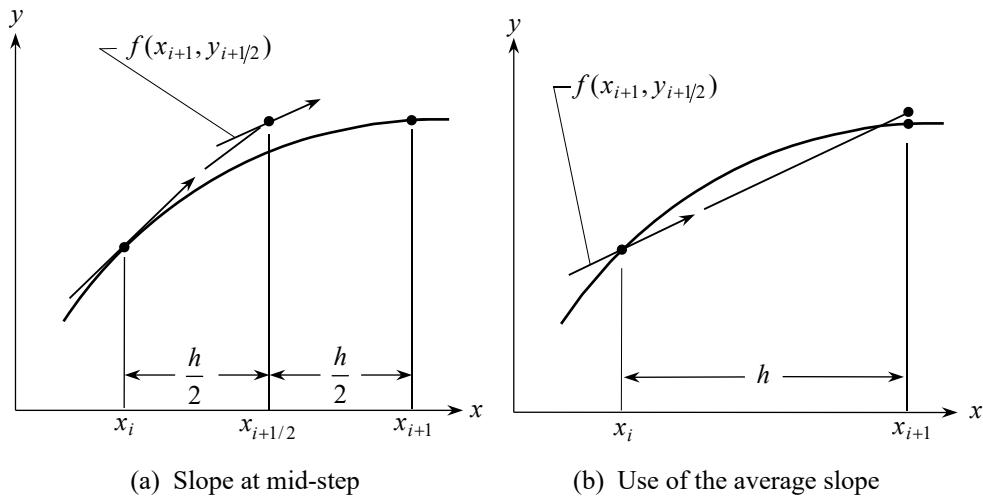


Figure 7.9 The modified Euler's method.

The idea of the modified Euler's method is similar to the Heun's method in the sense that accurate slope at the beginning of the step leads to an improved solution accuracy at the end of the step. The modified Euler's method determines the value and its slope at the mid-step as shown in Fig. 7.9(a). The computed slope is then used at the beginning of the step to predict the solution at the end of the step as illustrated in Fig. 7.9(b).

The solution at mid-step is first determined according to the Euler's method,

$$y_{i+1/2} = y_i + f(x_i, y_i) \frac{h}{2} \quad (7.30)$$

The slope at mid-step is then determined,

$$y'_{i+1/2} = f(x_{i+1/2}, y_{i+1/2}) \quad (7.31)$$

The computed slope is used at the beginning of the step to determine the solution at the end of the step from

$$y_{i+1} = y_i + f(x_{i+1/2}, y_{i+1/2}) h \quad (7.32)$$

Example 7.3 Use the modified Euler's method to solve the ordinary differential equation,

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0)=1$. Use the step size of $h=1.00$ in the computation. Then, develop a corresponding computer program to solve the problem again but by decreasing the step size to $h=0.25$. It is noted that the problem is identical to Examples 7.1 and 7.2 that were previously solved by using the Euler's and Heun's method, respectively.

For the first round with $h=1.00$ at $x_0=0$ and $y_0=1$, the solution at mid-step is determined from Eq. (7.30),

$$y_{1/2} = 1 + [(1)(1)](0.5) = 1.5 \quad (7.33a)$$

The slope at mid-step can then be determined by using Eq. (7.31),

$$y'_{1/2} = 1.5\cos(0.5) = 1.31637$$

The computed slope is used at the beginning of the step to determine the solution at the end of the step according to Eq. (7.32) as

$$y_1 = 1 + (1.31637)(1) = 2.31637 \quad (7.33b)$$

which has the exact error of

$$E_t = e^{\sin(1)} - 2.31637 = 0.00341 \quad (7.33c)$$

For the second round, $x_1 = 1$ and $y_1 = 2.31637$, the solution and its slope at mid-step, the computed solution at the end of the step and the exact error are

$$y_{1+1/2} = 2.31637 + [2.31637 \cos(1)](0.5) = 2.94214 \quad (7.34a)$$

$$y'_{1+1/2} = 2.94214 \cos(1.5) = 0.20812$$

$$y_2 = 2.31637 + (0.20812)(1) = 2.52449 \quad (7.34b)$$

$$E_t = e^{\sin(2)} - 2.52449 = -0.04191 \quad (7.34c)$$

Similarly, for the third round at $x_2 = 2$ with $y_2 = 2.52449$, the solution and its slope at mid-step, the computed solution at the end of the step and the exact error are

$$y_{2+1/2} = 2.52449 + [2.52449 \cos(2)](0.5) = 1.99921 \quad (7.35a)$$

$$y'_{2+1/2} = 1.99921 \cos(2.5) = -1.60165$$

$$y_3 = 2.52449 + (-1.60165)(1) = 0.92284 \quad (7.35b)$$

$$E_t = e^{\sin(3)} - 0.92284 = 0.22872 \quad (7.35c)$$

The solution obtained from the modified Euler's method using the step size of $h = 1$ is compared with the Euler's, Heun's and exact solutions as plotted in Fig. 7.10.

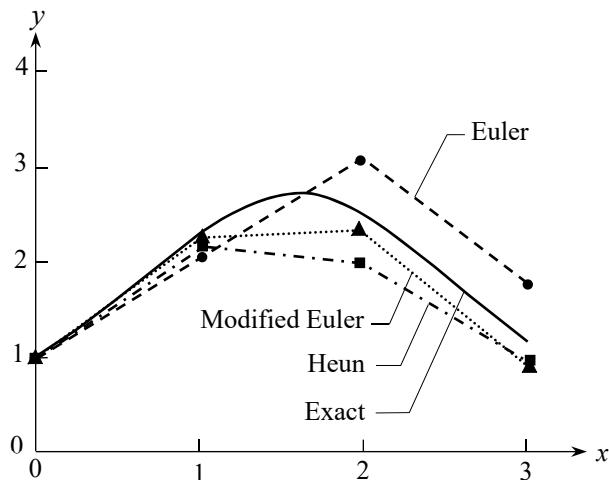


Figure 7.10 Comparison of the solutions obtained from the Euler's, Heun's and modified Euler's methods with the exact solution in Example 7.3.

Figure 7.11 shows the corresponding computer program of the modified Euler's method for solving the differential equation in Example 7.3. The computed solution using $h = 0.25$ is compared with the Euler's, Heun's and exact solutions in Table 7.2. The modified Euler's solution in the Table shows that the method provides the second-order solution accuracy, $O(h^2)$, in the same way as the Heun's method.

```
% Program Meuler
% A program for solving ordinary differential
% equation using the modified Euler's method
% Read initial conditions, number of steps,
% and step size:
%-----
func = @(x,y) y*cos(x);
%-----
x = input('\nEnter value of x: ');
y = input( 'Enter value of y: ');
n = input( 'Enter value of n: ');
h = input( 'Enter value of h: ');
fprintf('\nSolution with step size = %10.4e is:'
fprintf('\n      x           y');
for i = 1:n
    s0 = func(x,y);
    y1 = y + s0*h/2. ;
    x1 = x + h/2. ;
    sa = func(x1,y1);
    y = y + sa*h;
    x = x + h;
    fprintf('\n%16.6e%16.6e',x,y);
end
```

Figure 7.11 Computer program for solving the first-order ordinary differential equation by using the modified Euler's method in Example 7.3.

Table 7.2 Comparison of the solutions obtained from the Euler's, Heun's and modified Euler's methods with the exact solution.

Numerical solutions				
x	Exact solution	Euler	Heun	Modified Euler
0.00	1.00000	1.00000	1.00000	1.00000
0.25	1.28070	1.25000	1.27639	1.27906
0.50	1.61515	1.55279	1.60492	1.61263
0.75	1.97712	1.89346	1.95996	1.97545
1.00	2.31978	2.23982	2.29581	2.32096
1.25	2.58309	2.54236	2.55358	2.58805
1.50	2.71148	2.74278	2.67858	2.71888
1.75	2.67510	2.79128	2.64153	2.68173
2.00	2.48258	2.66690	2.45139	2.48539
2.25	2.17727	2.38944	2.15141	2.17541
2.50	1.81934	2.01419	1.80087	1.81444
2.75	1.46472	1.61078	1.45413	1.45952
3.00	1.15156	1.23857	1.14776	1.14820

7.5 Runge-Kutta Method

The Runge-Kutta method is popular and has been used to solve scientific and engineering problems that require high solution accuracy. The key idea of the method is to obtain a proper slope at x_i in order to determine accurate solution y_{i+1} at x_{i+1} . The predicted solution is first written in the form similar to the three methods previously explained,

$$y_{i+1} = y_i + \phi(x_i, y_i, h) h \quad (7.36)$$

where $\phi(x_i, y_i, h)$ is called the increment function. The increment function representing the average slope over the step h is expressed in a general form as

$$\phi = a_1 k_1 + a_2 k_2 + a_3 k_3 + \dots + a_n k_n \quad (7.37)$$

where $a_i, i = 1, 2, 3, \dots, n$ are constants and

$$k_1 = f(x_i, y_i) \quad (7.38a)$$

$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h) \quad (7.38b)$$

$$k_3 = f(x_i + p_2 h, y_i + q_{21} k_1 h + q_{22} k_2 h) \quad (7.38c)$$

$$\vdots \quad \vdots$$

$$k_n = f(x_i + p_{n-1} h, y_i + q_{n-1,1} k_1 h + q_{n-1,2} k_2 h + \dots + q_{n-1,n-1} k_{n-1} h) \quad (7.38n)$$

The subscript n in Eq. (7.37) denotes the order of the Runge-Kutta method. For example, the method is called the first-order Runge-Kutta method if $n = 1$. By considering Eqs. (7.36) - (7.38a), the first-order Runge-Kutta method is equivalent to the Euler method. If $n = 2$, Eqs. (7.36) - (7.38b) yield the second-order Runge-Kutta method. The parameters $k_i, i = 1, 2, 3, \dots, n$ in Eq. (7.38) depend on the given function on the right-hand side of the ordinary differential equation. The coefficients p and q are constants which can be determined and will be shown later. Determination of these coefficients for the fourth-order Runge-Kutta method is also presented in details in Appendix C. It is noted that the value for the parameter k_1 must be known prior to determining the parameter k_2 . Similarly, the value of the parameter k_2 must be known earlier in order to determine the parameter k_3 . To understand the Runge-Kutta method clearly, the following section explains the second-order Runge-Kutta method ($n = 2$) in details.

7.5.1 Second-order

For the second-order Runge-Kutta method ($n = 2$), the general form of Eqs. (7.36) - (7.38) become

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)h \quad (7.39)$$

where

$$k_1 = f(x_i, y_i) \quad (7.40)$$

and

$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h) \quad (7.41)$$

There are four unknowns of a_1 , a_2 , p_1 and q_{11} in Eqs. (7.39) - (7.41). These four unknowns are determined such that the second-order Runge-Kutta method provides the same solution accuracy as that obtained from the Taylor series expansion with three terms,

$$y_{i+1} = y_i + f(x_i, y_i)h + f'(x_i, y_i)\frac{h^2}{2} + \dots \quad (7.42)$$

From chain-rule, the first-order derivative term in Eq. (7.42) can be expressed as

$$f'(x_i, y_i) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \frac{dy}{dx} = \frac{\partial f}{\partial x} + f(x_i, y_i) \frac{\partial f}{\partial y} \quad (7.43)$$

By substituting Eq. (7.43) into Eq. (7.42),

$$y_{i+1} = y_i + f(x_i, y_i) h + \left(\frac{\partial f}{\partial x} + f(x_i, y_i) \frac{\partial f}{\partial y} \right) \frac{h^2}{2} + \dots \quad (7.44)$$

Thus, in order to determine the four unknowns, the second-order Runge-Kutta Eq. (7.39) must be written in the form of the Taylor series in Eq. (7.44). To do that, the Taylor series expansion with two variables

$$g(x+r, y+s) = g(x, y) + r \frac{\partial g}{\partial x} + s \frac{\partial g}{\partial y} \quad (7.45)$$

is first applied to Eq. (7.41) leading to

$$k_2 = f(x_i, y_i) + p_1 h \frac{\partial f}{\partial x} + q_{11} k_1 h \frac{\partial f}{\partial y} + \dots \quad (7.46)$$

By substituting k_1 from Eq. (7.40) into Eq. (7.46), then further substituting Eq. (7.46) into Eq. (7.39) and rearranging terms, Eq. (7.39) finally becomes

$$\begin{aligned} y_{i+1} &= y_i + [a_1 f(x_i, y_i) + a_2 f(x_i, y_i)] h \\ &\quad + \left[a_2 p_1 \frac{\partial f}{\partial x} + a_2 q_{11} f(x_i, y_i) \frac{\partial f}{\partial y} \right] h^2 + \dots \end{aligned} \quad (7.47)$$

By comparing the Runge-Kutta Eq. (7.47) with the Taylor series Eq. (7.44), the following three conditions are obtained,

$$a_1 + a_2 = 1 \quad (7.48a)$$

$$a_2 p_1 = 1/2 \quad (7.48b)$$

$$a_2 q_{11} = 1/2 \quad (7.48c)$$

Since there are four unknowns with the only three available conditions, an unknown must be given in order to determine the rest of the unknowns.

If $a_2 = 1/2$ is selected, then Eqs. (7.48a-c) yield

$$a_1 = 1/2 ; \quad p_1 = 1 ; \quad q_{11} = 1 \quad (7.49)$$

Then, the second-order Runge-Kutta Eq. (7.39) is

$$y_{i+1} = y_i + \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right) h \quad (7.50)$$

and Eqs. (7.40) - (7.41) are

$$k_1 = f(x_i, y_i) = \text{slope at the beginning of step } h$$

$$k_2 = f(x_i + h, y_i + h k_1) = \text{slope at the end of step } h$$

It is noted that Eq. (7.50) is identical to Eq. (7.26b) which is the Huen's method presented in Section 7.3.

If $a_2 = 1$ is selected, then Eqs. (7.48a-c) yield

$$a_1 = 0 \quad ; \quad p_1 = 1/2 \quad ; \quad q_{11} = 1/2 \quad (7.51)$$

Then, the second-order Runge-Kutta Eq. (7.39) is

$$y_{i+1} = y_i + (0 + k_2) h \quad (7.52)$$

and Eqs. (7.40) - (7.41) are

$$k_1 = f(x_i, y_i) = \text{slope at the beginning of step } h$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h k_1\right) = \text{slope at the end of step } h$$

It is also noted that Eq. (7.52) is identical to Eq. (7.32) which is the modified Euler's method presented in Section 7.4.

Equations (7.50) and (7.52) for the second-order Runge-Kutta method provide a solution with the second-order of accuracy, $O(h^2)$. The second-order Runge-Kutta method reduces the solution error to a quarter if the step size h is cut by a half.

7.5.2 Third-order

The third-order Runge-Kutta method ($n = 3$) can be obtained from the general expressions as shown in Eqs. (7.36) - (7.3.8). The widely used third-order Runge-Kutta method is in the following form,

$$y_{i+1} = y_i + \left[\frac{1}{6} (k_1 + 4 k_2 + k_3) \right] h \quad (7.53)$$

where

$$k_1 = f(x_i, y_i) \quad (7.54a)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h k_1\right) \quad (7.54b)$$

$$k_3 = f(x_i + h, y_i - h k_1 + 2h k_2) \quad (7.54c)$$

Solution error obtained from Eq. (7.53) decreases with the step size h as the third order, $O(h^3)$. The computational procedure of the third-order Runge-Kutta method is simple as shown in the following example.

Example 7.4 Develop a computer program by employing the third-order Runge-Kutta method to solve the differential equation

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0)=1$. Use the step size of $h = 0.25$ in the computation and compare the solution with the exact and Heun's solutions.

Starting from the initial condition of $y_0 = 1$ at $x_0 = 0$, Eqs. (7.53) - (7.54) give

$$\begin{aligned}
 k_1 &= (1)\cos(0) & = 1 \\
 k_2 &= \left[1 + \left(\frac{1}{2}\right)(0.25)(1)\right]\cos\left(\frac{0.25}{2}\right) & = 1.11622 \\
 k_3 &= [1 - (0.25)(1) + 2(0.25)(1.11622)]\cos(0.25) & = 1.26745 \\
 y_1 &= 1 + \left[\frac{1}{6}(1+4(1.11622)+1.26745)\right](0.25) & = 1.28051
 \end{aligned}$$

The solution of $y_1 = 1.28051$ obtained from the first step at $x_1 = 0.25$ is used to determine the solution at the end of the second step as

$$\begin{aligned}
 k_1 &= (1.28051)\cos(0.25) & = 1.24071 \\
 k_2 &= \left[1.28051 + \left(\frac{1}{2}\right)(0.25)(1.24071)\right]\cos\left(0.25 + \frac{0.25}{2}\right) & = 1.33584 \\
 k_3 &= [1.28051 - (0.25)(1.24071) + 2(0.25)(1.33584)]\cos(0.25 + 0.25) & = 1.43771 \\
 y_2 &= 1.28051 + \left[\frac{1}{6}(1.24071+4(1.33584)+1.43771)\right](0.25) & = 1.61475
 \end{aligned}$$

The same process is repeated to determine the solutions at the later steps. Figure 7.12 shows the computer programs of the third-order Runge-Kutta method for solving the differential equation in this example. The computed solutions at different time steps are compared with the exact and Heun's solutions as shown in Table 7.3.

```

% Program RK3
% A program for solving an ordinary
% differential equation by using the
% third-order Runge-Kutta method.
% Provide the f(x,y) function:
func = @(x,y)(y*cos(x));
% Read the initial conditions, number of
% steps and step size:
x(1) = input('
    'nEnter initial value of x: ');
y(1) = input('
    'Enter initial value of y: ');
n = input('
    'Enter number of steps: ');
h = input('
    'Enter the step size: ');
fprintf('\n
        x           y');
fprintf('\n%16.6e%16.6e',x(1),y(1));
for i = 1:n
    ak1 = func(x(i),y(i));
    xx = x(i) + h/2;
    yy = y(i) + h*ak1/2;
    ak2 = func(xx,yy);
    xx = x(i) + h;
    yy = y(i) - h*ak1 + 2.*h*ak2;
    ak3 = func(xx,yy);
    y(i+1) = y(i) + (ak1 + 4*ak2 + ak3)*h/6;
    x(i+1) = x(i) + h;
    fprintf('\n%16.6e%16.6e',x(i+1),y(i+1));
end
plot(x,y,'-or'), hold on
% Exact solution:
xe = 0:0.05:3; ye = exp(sin(xe));
plot(xe,ye,'-b')

```

Figure 7.12 Computer program for solving the first-order ordinary differential equation by using the third-order Runge-Kutta method in Example 7.4.

7.5.3 Fourth-order

The fourth-order Runge-Kutta method ($n = 4$) is widely used for solving many scientific and engineering problems by embedding itself in commercial software. The method provides solution with the fourth-order of accuracy, $O(h^4)$. The most popular form of the fourth-order Runge-Kutta method which was derived from the general form of Runge-Kutta Eqs. (7.36) - (7.38) is,

$$y_{i+1} = y_i + \left[\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\right]h \quad (7.55)$$

where

$$k_1 = f(x_i, y_i) \quad (7.56a)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h k_1\right) \quad (7.56b)$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h k_2\right) \quad (7.56c)$$

$$k_4 = f(x_i + h, y_i + h k_3) \quad (7.56d)$$

Details for the derivation of Eqs. (7.55) - (7.56a-d) are presented in Appendix C.

Example 7.5 Use the fourth-order Runge-Kutta method to solve the differential equation

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0)=1$ by developing a computer program. Then, employ the program to determine the solution by using the step size of $h = 0.25$. Compare the solution with the second-, third-order Runge-Kutta and exact solutions.

Details of the computational procedure for the first two steps are shown below. With the initial condition of $y_0 = 1$ at $x_0 = 0$, Eqs. (7.55) - (7.56) for the first step are

$$\begin{aligned} k_1 &= (1)\cos(0) &= 1 \\ k_2 &= \left[1 + \left(\frac{1}{2}\right)(0.25)(1)\right]\cos\left(\frac{0.25}{2}\right) &= 1.11622 \\ k_3 &= \left[1 + \left(\frac{1}{2}\right)(0.25)(1.11622)\right]\cos\left(\frac{0.25}{2}\right) &= 1.13064 \\ k_4 &= [1 + (0.25)(1.13064)]\cos(0.25) &= 1.24278 \\ y_1 &= 1 + \left[\frac{1}{6}(1+2(1.11622)+2(1.13064)+1.24278)\right](0.25) &= 1.28069 \end{aligned}$$

For the second step with the computed $y_1 = 1.28069$ at $x_1 = 0.25$, Eqs. (7.55) - (7.56) are

$$\begin{aligned} k_1 &= (1.28069)\cos(0.25) &= 1.24087 \\ k_2 &= \left[1.28069 + \left(\frac{1}{2}\right)(0.25)(1.24087)\right]\cos\left(0.25 + \frac{0.25}{2}\right) &= 1.33602 \\ k_3 &= \left[1.28069 + \left(\frac{1}{2}\right)(0.25)(1.33602)\right]\cos\left(0.25 + \frac{0.25}{2}\right) &= 1.34709 \\ k_4 &= [1.28069 + (0.25)(1.34709)]\cos(0.25 + 0.25) &= 1.41945 \\ y_2 &= 1.28069 + \left[\frac{1}{6}(1.24087 + 2(1.33602) + 2(1.34709) + 1.41945)\right](0.25) &= 1.61513 \end{aligned}$$

The process is repeated to determine solution for the other steps. Miscalculating a parameter in any step will cause error in the later steps. Thus, solving the problem by developing a computer program is essential. Figure 7.13 shows a fourth-order Runge-Kutta computer program for solving this problem.

User can employ the program to solve other differential equations by simply modifying the function declared in the program.

The computed solutions at different x are compared with the exact and the second- and third-order Runge-Kutta solutions in Table 7.3. The table shows that the fourth-order Runge-Kutta solutions are more accurate than those obtained from the second- and third-order Runge-Kutta methods. For example, at $x = 1.00$, the fourth-order Runge-Kutta solution is 2.31974. Such solution has only 0.002% error as compared to the exact solution of 2.31978. The solutions obtained from the Euler's, second- and third-order Runge-Kutta methods contain the errors of 3.45%, 1.03% and 0.02%, respectively. High solution accuracy of the fourth-order Runge-Kutta method has led to the method's popularity for solving many practical applications as well as using in academic research.

```
% Program RK4
% A program for solving an ordinary
% differential equation by using the
% fourth-order Runge-Kutta method.
% Provide the f(x,y) function:
func = @(x,y)(y*cos(x));
% Read the initial conditions, number of
% steps and step size:
x(1) = input( ...
    '\nEnter initial value of x: ');
y(1) = input( ...
    'Enter initial value of y: ');
n = input( 'Enter number of steps: ');
h = input( 'Enter the step size: ');
fprintf('\n      x           y');
fprintf('\n%16.6e%16.6e',x(1),y(1));
for i = 1:n
    ak1 = func(x(i),y(i));
    xx = x(i) + h/2.;
    yy = y(i) + h*ak1/2.;
    ak2 = func(xx,yy);
    yy = y(i) + h*ak2/2.;
    ak3 = func(xx,yy);
    xx = x(i) + h;
    yy = y(i) + h*ak3;
    ak4 = func(xx,yy);
    y(i+1) = y(i)+(ak1+2*ak2+2*ak3+ak4)*h/6;
    x(i+1) = x(i) + h;
    fprintf('\n%16.6e%16.6e',x(i+1),y(i+1));
end
plot(x,y,'-o'), hold on
% Exact solution:
xe = 0:.05:3; ye = exp(sin(xe));
plot(xe,ye,'-b')
```

Figure 7.13 Computer program for solving the first-order ordinary differential equation by using the fourth-order Runge-Kutta method in Example 7.5.

Table 7.3 Comparison of the second-, third- and fourth-order Runge-Kutta solutions obtained from solving the differential Eq. (7.13) with the exact solution.

Order of Runge-Kutta method				
<i>x</i>	Exact	Second	Third	Fourth
0.00	1.00000	1.00000	1.00000	1.00000
0.25	1.28070	1.27639	1.28051	1.28069
0.50	1.61515	1.60492	1.61475	1.61513
0.75	1.97712	1.95996	1.97657	1.97709
1.00	2.31978	2.29581	2.31923	2.31974
1.25	2.58309	2.55358	2.58273	2.58304
1.50	2.71148	2.67858	2.71149	2.71144
1.75	2.67510	2.64153	2.67555	2.67505
2.00	2.48258	2.45139	2.48342	2.48254
2.25	2.17727	2.15141	2.17832	2.17723
2.50	1.81934	1.80087	1.82034	1.81931
2.75	1.46472	1.45413	1.46547	1.46469
3.00	1.15156	1.14775	1.15201	1.15155

7.6 System of Equations

Practical scientific and engineering problems often require solving many first-order differential equations which are coupled. For example, the swinging pendulum problem explained in Section 7.1 consists of two first-order differential Eqs. (7.57a-b). The two differential equations are coupled and must be solved simultaneously for their solutions. In general, a system of n first-order differential equations can be written in the form,

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2, \dots, y_n) \quad (7.57a)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2, \dots, y_n) \quad (7.57b)$$

$$\vdots \qquad \vdots$$

$$\frac{dy_n}{dx} = f_n(x, y_1, y_2, \dots, y_n) \quad (7.57n)$$

The methods, such as the Euler's and Runge-Kutta methods explained earlier in Sections 7.2 - 7.5, can be modified to solve the above set of coupled first-order differential equations. The example below presents the use of the presented methods to solve a second-order differential equation which is separated into two coupled first-order differential equations.

Example 7.6 Employ the Euler's method to solve the second-order differential equation

$$\frac{d^2y}{dx^2} + 2\frac{dy}{dx} + 4y = 0 \quad (7.58)$$

with the initial conditions of $y(0)=2$ and $dy/dx(0)=0$. Determine the solution from $x=0$ to 3 by using the step sizes of $h=0.1$ and 0.01 .

The exact solution to the differential Eq. (7.58) with the given initial conditions is

$$y(x) = 2e^{-x} \left[\cos(\sqrt{3}x) + \frac{1}{\sqrt{3}} \sin(\sqrt{3}x) \right] \quad (7.59)$$

The Euler's method can be modified to solve the second-order differential Eq. (7.58). The differential equation is first separated into two first-order differential equations. For example, by assigning

$$\frac{dy}{dx} = z \quad (7.60a)$$

then, Eq. (7.58) becomes

$$\frac{dz}{dx} = -2z - 4y \quad (7.60b)$$

with the initial conditions of $y(x=0)=2$ and $z(x=0)=0$.

The Euler's Eq. (7.12) is then applied to Eqs. (7.60a-b) as follows

$$y_{i+1} = y_i + f_1(x_i, y_i, z_i) h = y_i + z_i h \quad (7.61a)$$

$$z_{i+1} = z_i + f_2(x_i, y_i, z_i) h = z_i + (-2z_i - 4y_i)h \quad (7.61b)$$

With the step size of $h = 0.1$, the first step yields

$$\begin{aligned} y_1 &= 2 + (0)(0.1) &= 2 \\ z_1 &= 0 + [-2(0) - 4(2)](0.1) &= -0.8 \end{aligned}$$

The computed solutions are then used in the second step to yield

$$\begin{aligned} y_2 &= 2 + (-0.8)(0.1) &= 1.92 \\ z_2 &= -0.8 + [-2(-0.8) - 4(2)](0.1) &= -1.44 \end{aligned}$$

The process is repeated until the last step at $x = 3$ is reached. Figure 7.14 shows the corresponding computer program that follows the above procedure for solving this example. The computed solutions by using the step sizes of $h = 0.1$ and 0.01 are compared with the exact solution as shown in Table 7.4.

```
% Program SysEul
% A program for solving a set of two
% ordinary first-order differential eqs.
% using the Euler's method.
% Provide the two functions:
func1 = @(x,y,z) (z);
func2 = @(x,y,z) (-2.*z - 4.*y);
% Read the initial conditions, number of
% steps and step size:
x(1) = input( ...
    '\nEnter initial value of x: ');
y(1) = input( ...
    '\nEnter initial value of y: ');
z(1) = input( ...
    '\nEnter initial value of z: ');
n = input( 'Enter number of steps: ');
h = input( 'Enter the step size: ');
fprintf('\n %s %15s %15s', 'x', 'y', 'z');

for i = 1:n
    f1 = func1(x(i),y(i),z(i));
    f2 = func2(x(i),y(i),z(i));
    y(i+1) = y(i) + f1*h;
    z(i+1) = z(i) + f2*h;
    x(i+1) = x(i) + h;
    fprintf('\n%16.6e%16.6e%16.6e', ...
        x(i+1),y(i+1),z(i+1));
end
plot(x,y,'-o'), hold on
% Exact solution:
xe = 0:.05:3; sq3 = sqrt(3.);
ye = 2*exp(-xe).* (cos(sq3*xe) + ...
    sin(sq3*xe)/sq3);
plot(xe, ye, '-b')
```

Figure 7.14 Computer program for solving the second-order ordinary differential equation by using the Euler's method in Example 7.6.

Example 7.7 Solve the problem in Example 7.6 again but by developing a computer program that employs the fourth-order Runge-Kutta method. Use the step size of $h = 0.1$ and compare the solution obtained with the exact and the Euler's solutions.

The second-order differential equation in Eq. (7.58) is firstly separated into the two first-order differential equations as shown in Eqs. (7.60a-b). The fourth-order Runge-Kutta method as shown in Eqs. (7.55) - (7.56) is then applied to each differential equation as follows,

$$y_{i+1} = y_i + \left[\frac{1}{6} (k_{1y} + 2k_{2y} + 2k_{3y} + k_{4y}) \right] h \quad (7.62a)$$

$$z_{i+1} = z_i + \left[\frac{1}{6} (k_{1z} + 2k_{2z} + 2k_{3z} + k_{4z}) \right] h \quad (7.62b)$$

where

$$k_{1y} = f_1(x_i, y_i, z_i) \quad (7.63a)$$

$$k_{2y} = f_1\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_{1y}, z_i + \frac{1}{2}hk_{1z}\right) \quad (7.63b)$$

$$k_{3y} = f_1\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_{2y}, z_i + \frac{1}{2}hk_{2z}\right) \quad (7.63c)$$

$$k_{4y} = f_1(x_i + h, y_i + hk_{3y}, z_i + hk_{3z}) \quad (7.63d)$$

$$k_{1z} = f_2(x_i, y_i, z_i) \quad (7.64a)$$

$$k_{2z} = f_2\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_{1y}, z_i + \frac{1}{2}hk_{1z}\right) \quad (7.64b)$$

$$k_{3z} = f_2\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_{2y}, z_i + \frac{1}{2}hk_{2z}\right) \quad (7.64c)$$

$$k_{4z} = f_2(x_i + h, y_i + hk_{3y}, z_i + hk_{3z}) \quad (7.64d)$$

The functions f_1 and f_2 are

$$f_1(x_i, y_i, z_i) = z_i \quad \text{and} \quad f_2(x_i, y_i, z_i) = -2z_i - 4y_i$$

For example, at the first step, $x_0 = 0$, $y_0 = 2$, $z_0 = 0$ where $h = 0.1$, Eqs. (7.63) - (7.64) are

$$k_{1y} = f_1(0, 2, 0) = 0$$

$$k_{1z} = f_2(0, 2, 0) = -8$$

$$k_{2y} = f_1(0.05, 2, -0.4) = -0.4$$

$$k_{2z} = f_2(0.05, 2, -0.4) = -7.2$$

$$k_{3y} = f_1(0.05, 1.98, -0.36) = -0.36$$

$$k_{3z} = f_2(0.05, 1.98, -0.36) = -7.2$$

$$k_{4y} = f_1(0.10, 1.964, -0.72) = -0.72$$

$$k_{4z} = f_2(0.10, 1.964, -0.72) = -6.416$$

Then, the new solutions of y and z in Eqs. (7.62a-b) are

$$y_1 = 2 + \left[\frac{1}{6}(0 + 2(-0.4) + 2(-0.36) + (-0.72)) \right](0.1) = 1.962667$$

$$z_1 = 0 + \left[\frac{1}{6}(-8 + 2(-7.2) + 2(-7.2) + (-6.416)) \right](0.1) = -0.720267$$

These solutions are used in the computation of the second step. The process is repeated until the last step is reached. Figure 7.5 shows the corresponding computer program for solving this example by using the fourth-order Runge-Kutta method. The computed solution is compared with the exact and Euler's solutions in Table 7.4.

Table 7.4 highlights the solution accuracy obtained from the fourth-order Runge-Kutta method. The table compares the Runge-Kutta solution with the Euler's solutions that use different step sizes. For example, the exact solution at $x = 1.0$ is 0.301149. By using the step size of $h = 0.1$, the Runge-Kutta solution has the error only 0.004% while the Euler's solution produces the error 38%. The Euler's method still yields the error 3.6% even though the step size is reduced to one-tenth with $h = 0.01$. Since the fourth-order Runge-Kutta method can provide high solution accuracy and a corresponding computer program can be easily developed, the method is widely used for solving ordinary differential equations.

```
% Program SysRK4
% A program for solving a set of two
% ordinary first-order differential eqs.
% using fourth-order Runge-Kutta method.
% Provide the two functions:
func1 = @(x,y,z)(z);
func2 = @(x,y,z)(-2.*z - 4.*y);
% Read the initial conditions, number of
% steps and step size:
x(1) = input( ...
    'nEnter initial value of x: ');
y(1) = input( ...
    'Enter initial value of y: ');
z(1) = input( ...
    'Enter initial value of z: ');
n = input( 'Enter number of steps: ');
h = input( 'Enter the step size: ');
fprintf('n%9s %15s %15s','x','y','z');
fprintf('n%16.6e%16.6e%16.6e', ...
    x(1),y(1),z(1));
for i = 1:n
    akly = func1(x(i),y(i),z(i));
    aklz = func2(x(i),y(i),z(i));
    xx = x(i) + h/2;
    yy = y(i) + h*akly/2;
    zz = z(i) + h*aklz/2.;

    ak2y = func1(xx,yy,zz);
    ak2z = func2(xx,yy,zz);
    yy = y(i) + h*ak2y/2.;
    zz = z(i) + h*ak2z/2.;

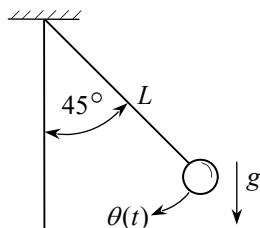
    ak3y = func1(xx,yy,zz);
    ak3z = func2(xx,yy,zz);
    xx = x(i) + h;
    yy = y(i) + h*ak3y;
    zz = z(i) + h*ak3z;
    ak4y = func1(xx,yy,zz);
    ak4z = func2(xx,yy,zz);
    y(i+1) = y(i) + (akly + 2*ak2y + ...
        2*ak3y + ak4y)*h/6;
    z(i+1) = z(i) + (aklz + 2*ak2z + ...
        2*ak3z + ak4z)*h/6;
    x(i+1) = x(i) + h;
    fprintf('n%16.6e%16.6e%16.6e', ...
        x(i+1),y(i+1),z(i+1));
end
plot(x,y,'-o'), hold on
% Exact solution:
xe = 0:.05:3; sq3 = sqrt(3.);
ye = 2*exp(-xe).* (cos(sq3*xe) + ...
    sin(sq3*xe)/sq3);
plot(xe,ye,'-b')
```

Figure 7.15 Computer program for solving the second-order ordinary differential equation by using the fourth-order Runge-Kutta method in Example 7.7.

Table 7.4 Comparison of the Euler's and fourth-order Runge-Kutta solutions obtained from solving examples 7.6 and 7.7 with the exact solution.

x	Exact	Euler		Runge-Kutta
		<i>h</i> = 0.1	<i>h</i> = 0.01	<i>h</i> = 0.1
0.00	2.000000	2.000000	2.000000	2.000000
0.50	1.319400	1.359360	1.322049	1.319407
1.00	0.301149	0.185380	0.290313	0.301136
1.50	-0.248709	-0.429154	-0.264388	-0.248732
2.00	-0.306246	-0.400115	-0.314619	-0.306259
2.50	-0.149181	-0.121281	-0.147723	-0.149181
3.00	-0.004579	0.076168	0.001437	-0.004571

Example 7.8 Use the Euler's and fourth-order Runge-Kutta methods to solve the swinging pendulum angle $\theta(t)$ from the second-order differential equation



$$\frac{d^2\theta}{dt^2} + \frac{g}{L} \sin \theta = 0 \quad (7.2)$$

Use the gravitational accelerating constant of $g = 9.8 \text{ m/sec}^2$ and the chord length of $L = 0.5 \text{ m}$. The pendulum is released from the angle of $\theta_0 = \pi/4$ radian.

As mentioned earlier in section 7.1, the governing differential Eq. (7.2) is nonlinear. Derivation of the exact solution is lengthy

Figure 7.16 Swinging pendulum.

and difficult. For small swinging angle, the governing differential equation becomes linear and the exact solution can be derived easily as shown by Eqs. (7.5) - (7.9). The exact solution for small swinging angle is

$$\theta(t) = \theta_0 \cos \sqrt{\frac{g}{L}} t \quad (7.8)$$

By substituting the given values θ_0 , g and L , the solution is

$$\theta(t) = \frac{\pi}{4} \cos \sqrt{\frac{9.8}{0.5}} t \quad (7.65)$$

as shown in Table 7.5.

Approximate solution to the nonlinear differential Eq. (7.2) can be obtained conveniently by applying the Euler's or Rung-Kutta method. The procedure starts from separating the governing second-order differential equation into two first-order differential equations as follows,

$$\frac{d\theta}{dt} = \beta \quad (7.3a)$$

$$\frac{d\beta}{dt} = -\frac{g}{L} \sin \theta \quad (7.3b)$$

The Euler's or the Runge-Kutta method can then be applied to solve these two equations simultaneously by using the same procedure as explained in Examples 7.6 and 7.7. Corresponding computer programs similar to those shown in Figs. 7.14 and 7.15 can also developed for their solutions. Table 7.5 shows the comparison of the solutions obtained from the two methods by using different step sizes. The table also shows a relatively large difference between the solutions arisen from solving the linear and nonlinear differential equations. The solution difference suggests the importance for obtaining the solution of the original nonlinear equation. This is because nonlinear differential equations always occur in practical scientific and engineering problems.

Table 7.5 Comparison of the solutions for the swinging pendulum in Example 7.8 at different times.

Nonlinear solution (θ , radian)				
Time t (sec)	Linear solution (θ , radian)	Euler $h=0.01$	Euler $h=0.001$	Runge-Kutta $h=0.01$
0.0	0.785398	0.785398	0.785398	0.785398
0.2	0.497118	0.531681	0.522414	0.521418
0.4	-0.156096	-0.106033	-0.104951	-0.104747
0.6	-0.694720	-0.691733	-0.659939	-0.656418
0.8	-0.723351	-0.820165	-0.764768	-0.758862
1.0	-0.220971	-0.398713	-0.353641	-0.349142

7.7 MATLAB Commands

MATLAB has several commands for solving ordinary differential equations. These commands include `ode45`, `ode23`, `ode113`, etc. The format for using these commands is

```
[x,y] = solver(odefunc,span,y0)
```

where `solver` is the command used, such as `ode45`, `ode23` or `ode113`.
 `odefunc` is the function on to be solved
 `span` is the interval and step size
 `y0` is the initial condition

Example 7.9 Employ the MATLAB command `ode45` to solve the differential equation

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0)=1$. Use the step size of 0.25 for the time interval of $x=0$ to $x=3$.

The MATLAB commands for solving the differential Eq. (7.13) and its solution are as follows

```
>> f = inline('y*cos(x)', 'x', 'y');
>> [x,y] = ode45(f, [0:0.25:3], 1);
>> y

y =
1.0000
1.2807
1.6151
1.9771
2.3198
2.5831
2.7115
2.6751
2.4826
2.1773
1.8193
1.4647
1.1516
```

The solutions above are quite accurate as compared to the exact solution. Details of the MATLAB commands and their solution accuracy are provided in Table 7.6.

Table 7.6 Details of the commands `ode23`, `ode45` and `ode113` for solving a first-order ordinary differential equation.

Command	Accuracy	Details
<code>ode23</code>	Low	Use the second- and third-order Runge-Kutta methods.
<code>ode45</code>	Medium	Use the fourth- and fifth-order Runge-Kutta methods. The command is popular and widely used.
<code>ode113</code>	Medium to high	Use the multistep methods of Adams-Bashforth and Adams-Moulton formula explained in section 7.8.

Example 7.10 Employ a MATLAB command `ode45` to solve the set of two first-order differential equations in Example 7.6

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = -2y_2 - 4y_1$$

with the initial conditions of $y_1(x=0)=2$ and $y_2(x=0)=0$.

An m-file under the name `sysode.m` corresponding to the given differential equations is firstly established as follows

```
function dy = sysode(x,y)
dy = zeros(2,1);
dy(1) = y(2);
dy(2) = -2*y(2) - 4*y(1);
```

Then, the MATLAB command `ode45` is employed by using the format

```
>> [x,y] = ode45(@sysode,[0:0.5:3],[2 0])
```

The response from MATLAB is as follows

```
x =
0
0.5000
1.0000
1.5000
2.0000
2.5000
3.0000

y =
2.0000      0
1.3194   -2.1339
0.3012   -1.6772
-0.2487   -0.5331
-0.3063    0.1981
-0.1492    0.3518
-0.0046    0.2036
```

The y -solution above consists of two columns. The first column (y_1) is the solution to the problem while the second column contains the values of y_2 , i.e., derivatives of y_1 . The solution (y_1) to the problem is accurate as compared to the exact solution in Table 7.4.

7.8 Multistep Methods

All methods presented in Sections 7.2 - 7.6 are for determining a new solution y_{i+1} at x_{i+1} from the computed solution y_i at x_i as depicted in Fig. 7.17(a). These methods are called *one-step method* because only the computed solution from the previous step is used to determine a new solution at the next step. Since many solutions have been computed, they can be used to determine a new solution as described by Fig. 7.17(b). The new solution should be more accurate because it was determined from many previous computed solutions. Such concept has led the so called *multistep method* as will be presented in this section.

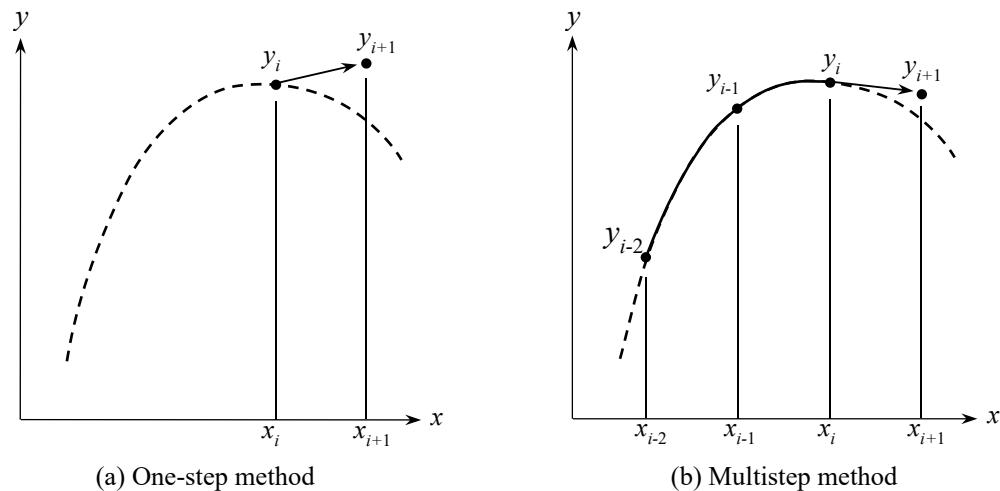


Figure 7.17 Concept of the one- and multistep methods.

7.8.1 Non-self-starting Heun's method

One of the simple multistep methods is based on the Heun's method studied in section 7.3. The Heun's method consists of two steps for determining the predictor and corrector as shown in Eqs. (7.26a-b). The predictor y_{t+1}^0 is determined from

$$y_{i+1}^0 = y_i + f(x_i, y_i) h \quad (7.26a)$$

where $f(x_i, y_i)$ is the slope at x_i as shown in Eq. (7.18a). The computed y_{i+1}^0 from Eq. (7.26a) is used to determine the slope at x_{i+1} . Then, the corrector is determined from

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} h \quad (7.26b)$$

The above procedure suggests that if a more accurate predictor y_{i+1}^0 can be determined, the computed solution accuracy is increased. A more accurate predictor y_{i+1}^0 can be obtained by following the concept of Fig. 7.18(b). The slope at x_i for determining the solution y_{i+1} is computed from the solution at x_{i-1} with the step size of $2h$ as

$$y_{i+1}^0 = y_{i-1} + f(x_i, y_i) 2h \quad (7.66)$$

Improved solution accuracy obtained from the predictor in Eq. (7.66) will be demonstrated in Example 7.11. It is noted that, at the very beginning of the computational process for determining y_1 , only y_0 is available while y_{-1} is not known. Such incomplete information at the starting point of the computation, has led to the name of the *non-self-starting Heun's method*.

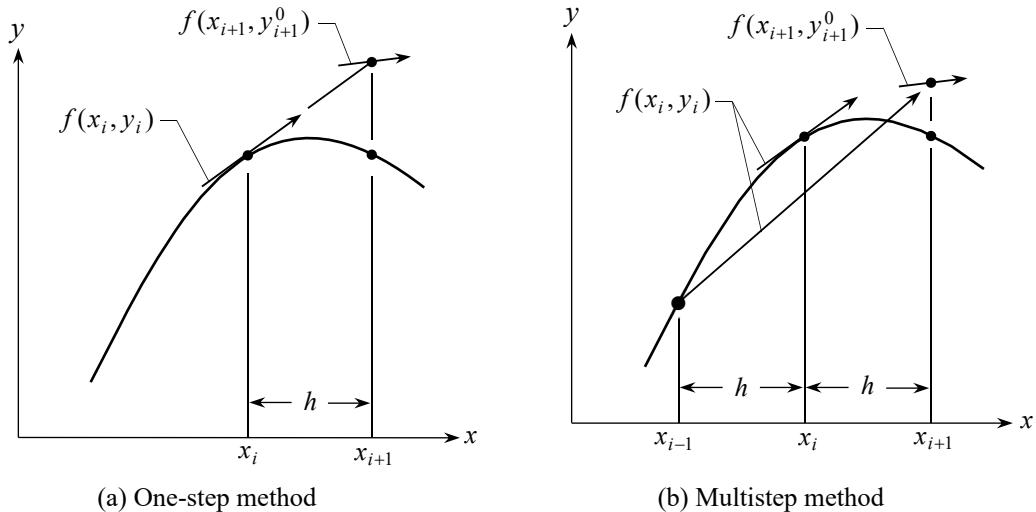


Figure 7.18 Concept of the one- and multistep non-self-starting Heun's methods.

In conclusion, the non-self-starting Heun's method is the predictor-corrector method that consists of two steps:

$$\text{Predictor: } y_{i+1}^0 = y_{i-1} + f(x_i, y_i) 2h \quad (7.67a)$$

$$\text{Corrector: } y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2} h \quad (7.67b)$$

Example 7.11 Employ the non-self-starting Heun's method to solve the differential equation

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0)=1$. Use the step size of $h = 1.00$ and compare the computed solution with those obtained from the one-step Heun's method in Example 7.2.

It is noted that the non-self-starting Heun's method requires the value y_{-1} at $x = -h = -1$ to determine the solution at the first step. The value y_{-1} may be determined from the fourth-order Runge-Kutta method as described in section 7.5.3 or by employing the computer program in Fig. 7.13. The method yields $y_{-1} = 0.43036$. The predictor in Eq. (7.67a) for the first step of the computation is

$$y_1^0 = 0.43036 + [(1)(1)](2)(1) = 2.43036 \quad (7.68a)$$

Then, the slope at the end of the step is

$$f(x_1, y_1^0) = (2.43036) \cos(1) = 1.31313$$

and the corrector in Eq. (7.67b) gives the solution

$$y_1 = 1 + \frac{1+1.31313}{2}(1) = 2.15657 \quad (7.68b)$$

The true error is

$$E_t = e^{\sin(1)} - 2.15657 = 0.16321 \quad (7.68c)$$

which is less than the error of 0.27948 obtained from the one-step Heun's method as shown in Eq. (7.27c).

Similarly, the second step of the computation starts from $x_1 = 1$ and $y_1 = 2.15657$. The predictor, slope at end of the step, corrector and true error are

$$y_2^0 = 1 + [2.15657 \cos(1)](2)(1) = 3.33040 \quad (7.69a)$$

$$f(x_2, y_2^0) = (3.33040) \cos(2) = -1.38594$$

$$y_2 = 2.15657 + \frac{1.16520 - 1.38594}{2}(1) = 2.04620 \quad (7.69b)$$

$$E_t = e^{\sin(2)} - 2.04620 = 0.43638 \quad (7.69c)$$

Again, the solution error produced by the non-self-starting Heun's method is less than that from the one-step Heun's method as shown in Eq. (7.28c).

7.8.2 Adams-Bashforth method

One of the multistep methods widely used and implemented in many commercial software is based on the Adams-Bashforth formulas. The formulas are sometimes called the Adams *open* formulas. The formulas were derived from the Taylor series expansion as shown in Eq. (6.134)

$$\begin{aligned} y_{i+1} &= y_i + f_i h + \frac{f'_i}{2!} h^2 + \frac{f''_i}{3!} h^3 + \dots \\ \text{or } y_{i+1} &= y_i + h \left[f_i + \frac{h}{2} f'_i + \frac{h^2}{6} f''_i + \dots \right] \end{aligned} \quad (7.70)$$

For example, the first backward divided-difference as shown in Eq. (6.138) is

$$f'_i = \frac{f_i - f_{i-1}}{h} + \frac{h}{2} f''_i + O(h^2) \quad (7.71)$$

By substituting Eq. (7.71) into Eq. (7.70)

$$y_{i+1} = y_i + h \left[f_i + \frac{h}{2} \left(\frac{f_i - f_{i-1}}{h} + \frac{h}{2} f''_i + O(h^2) \right) + \frac{h^2}{6} f''_i + \dots \right]$$

and arranging terms to get

$$y_{i+1} = y_i + h \left(\frac{3}{2} f_i - \frac{1}{2} f_{i-1} \right) + \frac{5}{12} h^3 f''_i + O(h^4)$$

$$\text{or } y_{i+1} = y_i + h \left(\frac{3}{2} f_i - \frac{1}{2} f_{i-1} \right) + O(h^3) \quad (7.72)$$

Equation (7.72) is called the second-order Adams open formula. It is called the open formula because the new solution y_{i+1} can be determined directly from the known functions f_i and f_{i-1} previously computed in the earlier steps. It should be noted that for the first step at $x = 0$, y_i and f_i are known and

y_{i-1} may be determined by using the Runge-Kutta method as explained in Example 7.11. Application of the Adams open formula is presented in Example 7.12.

Similarly, the third-order Adams open formula can be derived from the second backward divided-difference from Table 6.5

$$f_i'' = \frac{f_i - 2f_{i-1} + f_{i-2}}{h^2} + O(h) \quad (7.73)$$

By substituting Eqs. (7.71) and (7.73) into Eq. (7.70) and arranging terms, the new solution y_{i+1} is

$$y_{i+1} = y_i + h \left(\frac{23}{12} f_i - \frac{16}{12} f_{i-1} + \frac{5}{12} f_{i-2} \right) + O(h^4) \quad (7.74)$$

which is called the third-order open Adams formula. High-order open Adams-Basforth formulas can be derived by using the same procedure. The new solution y_{i+1} can be written in a more general form as

$$y_{i+1} = y_i + h \sum_{k=0}^{n-1} \beta_k f_{i-k} + O(h^{n+1}) \quad (7.75)$$

where n is the order of the Adams-Basforth formula and the coefficients β_k are shown in Table 7.7.

Table 7.7 Coefficients β_k in the Adams-Basforth formula.

Order	β_0	β_1	β_2	β_3	β_4	β_5
1	1					
2	$\frac{3}{2}$	$-\frac{1}{2}$				
3	$\frac{23}{12}$	$-\frac{16}{12}$	$\frac{5}{12}$			
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{9}{24}$		
5	$\frac{1,901}{720}$	$-\frac{2,774}{720}$	$\frac{2,616}{720}$	$-\frac{1,274}{720}$	$\frac{251}{720}$	
6	$\frac{4,277}{1,440}$	$-\frac{7,923}{1,440}$	$\frac{9,982}{1,440}$	$-\frac{7,298}{1,440}$	$\frac{2,877}{1,440}$	$-\frac{475}{1,440}$

Example 7.12 Develop a computer program by using the fourth-order Adams-Basforth formula to solve the differential equation

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0)=1$. Use the step size of $h = 0.25$ and compare the solution obtained with the exact solution.

From Eq. (7.75) and Table 7.7, the fourth-order Adams-Basforth formula is

$$y_{i+1} = y_i + h \left(\frac{55}{24} f_i - \frac{59}{24} f_{i-1} + \frac{37}{24} f_{i-2} - \frac{9}{24} f_{i-3} \right) \quad (7.76)$$

For the first step $i = 0$ of the computation at $x_0 = 0$, the given initial condition is $y_0 = 1$. However, Eq. (7.76) in the formula needs the values y_{-1} , y_{-2} , y_{-3} in order to determine f_{-1} , f_{-2} and f_{-3} , respectively. The fourth-order Runge-Kutta computer program as shown in Fig. 7.13 can be used to yield $y_{-1} = 0.78083$, $y_{-2} = 0.61914$ and $y_{-3} = 0.50579$. Thus, the values of the functions in Eq. (7.76) when $i = 0$ are

$$\begin{aligned}f_0 &= (1) \cos(0) &= 1.00000 \\f_{-1} &= (0.78083) \cos(-0.25) &= 0.75656 \\f_{-2} &= (0.61914) \cos(-0.50) &= 0.54335 \\f_{-3} &= (0.50579) \cos(-0.75) &= 0.37008\end{aligned}$$

Thus, the solution y_1 at the first step by using Eq. (7.76) is

$$\begin{aligned}y_1 &= 1 + 0.25 \left(\frac{55}{24}(1.00000) - \frac{59}{24}(0.75656) + \frac{37}{24}(0.54335) - \frac{9}{24}(0.37008) \right) \\&= 1.28267\end{aligned}$$

The solution obtained from the first step is used to determine the solution at the second step. The same process is repeated for determining the solutions at the later steps. The process is terminated when the specified number of steps is met or the last step is reached. A corresponding computer program that uses the fourth-order Adams-Bashforth formula is shown in Fig. 7.19. The values y_{-1} , y_{-2} , y_{-3} required by the program can be determined by using the fourth-order Runge-Kutta computer program as shown in Fig. 7.13. The computed solution obtained from the fourth-order Adams-Bashforth computer program is compared with the exact solution as shown in Table 7.9.

```
% Program AdamBas
% A program for solving an ordinary
% differential equation by using the
% fourth-order Adams-Bashforth method.
% Provide the f(x,y) function:
func = @(x,y) (y*cos(x));
% Read the initial conditions of x0, y0,
% y-1, y-2, y-3, the number of step and
% step size:
x(1) = input( ...
    '\nEnter initial value of x: ');
y(1) = input( ...
    '\nEnter initial value of y: ');
ym1 = input( ...
    '\nEnter initial value of y-1: ');
ym2 = input( ...
    '\nEnter initial value of y-2: ');
ym3 = input( ...
    '\nEnter initial value of y-3: ');
n = input( 'Enter number of steps: ');
h = input( 'Enter the step size: ');

fprintf('\n %9s %15s','x','y');
for i = 1:n
    f0 = func(x(i),y(i));
    xm1 = x(i) - h;
    f1 = func(xm1,ym1);
    xm2 = x(i) - 2.*h;
    f2 = func(xm2,ym2);
    xm3 = x(i) - 3.*h;
    f3 = func(xm3,ym3);
    ym3 = ym2;
    ym2 = ym1;
    ym1 = y(i);
    y(i+1) = y(i) + (55*f0 - 59*f1 + ...
        37*f2 - 9*f3)*h/24;
    x(i+1) = x(i) + h;
    fprintf('\n%16.6e%16.6e',x(i),y(i));
end
% Exact solution:
plot(x,y,'-or'), hold on
xe = 0:.05:3; ye = exp(sin(xe));
plot(xe,ye,'b')
```

Figure 7.19 Computer program for solving the first-order ordinary differential equation by using the fourth-order open Adams-Bashforth formula in Example 7.12.

7.8.3 Adams-Moulton method

The Adams-Moulton method is similar to the Adams-Bashforth method but can provide higher solution accuracy for general problems. The Adams-Moulton formulas are derived by using the backward Taylor series expansion in the form

$$\begin{aligned} y_i &= y_{i+1} - f_{i+1} h + \frac{f'_{i+1}}{2!} h^2 - \frac{f''_{i+1}}{3!} h^3 + \dots \\ \text{or } y_{i+1} &= y_i + h \left[f_{i+1} - \frac{h}{2!} f'_{i+1} + \frac{h^2}{3!} f''_{i+1} - \dots \right] \end{aligned} \quad (7.77)$$

The first-order derivative term in Eq. (7.77) is approximate by using the backward divided-difference as

$$f'_{i+1} = \frac{f_{i+1} - f_i}{h} + \frac{h}{2} f''_{i+1} + O(h^2) \quad (7.78)$$

By substituting Eq. (7.78) into Eq. (7.77) and arranging terms to yield

$$y_{i+1} = y_i + h \left(\frac{1}{2} f_{i+1} + \frac{1}{2} f_i \right) + O(h^3) \quad (7.79)$$

Equation (7.79) is called the second-order Adams closed formula. It is called the closed formula because the function f_{i+1} on the right-hand side of Eq. (7.79) depends on the value y_{i+1} which is also unknown.

Other high-order Adams closed formulas can be derived by using the same procedure as explained above. Details of the derivation are omitted herein and are left as exercise. The Adams-Moulton formulas can be written in a general form as follow

$$y_{i+1} = y_i + h \sum_{k=0}^{n-1} \beta_k f_{i-k+1} + O(h^{n+1}) \quad (7.80)$$

where n is the order of the formula and β_k are the coefficients as shown in Table 7.8.

Table 7.8 Coefficients β_k in the Adams-Moulton formulas.

Order	β_0	β_1	β_2	β_3	β_4	β_5
1	1					
2	$\frac{1}{2}$	$\frac{1}{2}$				
3	$\frac{5}{12}$	$\frac{8}{12}$	$-\frac{1}{12}$			
4	$\frac{9}{24}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$		
5	$\frac{251}{720}$	$\frac{646}{720}$	$-\frac{264}{720}$	$\frac{106}{720}$	$-\frac{19}{720}$	
6	$\frac{475}{1,440}$	$\frac{1,427}{1,440}$	$-\frac{798}{1,440}$	$\frac{482}{1,440}$	$-\frac{173}{1,440}$	$\frac{27}{1,440}$

Example 7.13 Apply the fourth-order Adams-Moulton formula to solve the differential equation

$$\frac{dy}{dx} = y \cos x \quad (7.13)$$

with the initial condition of $y(0)=1$. Use the step size of $h = 0.25$ and compare the solution obtained with the exact solution and the solution from the fourth-order Adams-Bashforth formula in Example 7.12.

From Eq. (7.80) and Table 7.8, the fourth-order Adams-Moulton formula is

$$y_{i+1} = y_i + h \left(\frac{9}{24} f_{i+1} + \frac{19}{24} f_i - \frac{5}{24} f_{i-1} + \frac{1}{24} f_{i-2} \right) \quad (7.81)$$

Similar to the Adams-Bashforth method in Example 7.12, the initial condition at the first step $x_0 = 0$ is $y_0 = 1$. The fourth-order Adams-Moulton formula in Eq. (7.81) needs f_{-1} and f_{-2} which are determined from y_{-1} and y_{-2} , respectively. These values y_{-1} and y_{-2} can be determined by using the fourth-order Runge-Kutta computer program in Fig. 7.13 to yield $y_{-1} = 0.78083$ and $y_{-2} = 0.61914$. The given y_0 and the computed y_{-1} and y_{-2} lead to the three function values of $f_0 = 1.00000$, $f_{-1} = 0.75656$, $f_{-2} = 0.54335$. Thus, Eq. (7.81) for the first step is

$$y_1 = 1 + 0.25 \left[\frac{9}{24} f_1 + \frac{19}{24} (1.00000) - \frac{5}{24} (0.75656) + \frac{1}{24} (0.54335) \right] \quad (7.82)$$

It is noted that, by using the given function

$$f_1 = y_1 \cos(x_1) = y_1 \cos(0.25) = 0.96891 y_1$$

thus, the solution of y_1 from Eq. (7.82) is

$$y_1 = 1.28049$$

which is more accurate than that from the Adams-Bashforth formula as compared to the exact solution. Table 7.9 compares the Adams-Bashforth and Adams-Moulton solutions with the exact solution. The table also shows the true errors produced by the two methods.

Table 7.9 Comparison of the Adams-Bashforth and Adams-Moulton solutions with the exact solution for Example 7.13.

x	Adams-Bashforth		Adams-Moulton		
	Exact solution	Solution	Error	Solution	Error
0.00	1.00000	1.00000	0.00000	1.00000	0.00000
0.25	1.28070	1.28267	-0.00197	1.28049	0.00021
0.50	1.61515	1.62075	-0.00056	1.61468	0.00047
0.75	1.97712	1.98633	-0.00921	1.97651	0.00061
1.00	2.31978	2.33009	-0.01031	2.31934	0.00044
1.25	2.58309	2.58982	-0.00673	2.58316	-0.00007
1.50	2.71148	2.71076	0.00072	2.71215	-0.00067
1.75	2.67510	2.66770	0.00740	2.67603	-0.00093
2.00	2.48258	2.47414	0.00844	2.48324	-0.00066
2.25	2.17727	2.17385	0.00342	2.17733	-0.00006
2.50	1.81934	1.82303	-0.00369	1.81887	0.00047
2.75	1.46472	1.47328	-0.00856	1.46408	0.00064
3.00	1.15156	1.16093	-0.00937	1.15106	0.00050

7.9 Closure

Popular methods for solving ordinary differential equations are presented in this chapter. These methods are the Euler's, Heun's, modified Euler's and Runge-Kutta methods. The methods can be applied to solve higher-order differential equations or a set of first-order differential equations. Among these methods, the Euler's method is considered as the simplest one. The method is easy to understand and can be applied to solve both linear and nonlinear ordinary differential equations.

The drawback of the Euler's method is that it does not produce solution with high accuracy, especially when using with a large step size. The Heun's and modified Euler's methods provide solutions with higher accuracy by determining proper slope within the step that is used to determine the new solution. Such idea for determining more accurate slope within the step is used as a basis in the development of the Runge-Kutta method. The Runge-Kutta method provides proper slope within the time step to further produce a more accurate solution. The proper slope is determined from matching the coefficients in the Runge-Kutta equation with the Taylor series expansion. The fourth Runge-Kutta method is widely used and implemented in many commercial software for solving ordinary differential equations.

The methods mentioned above are called the one-step method because the only solution computed at the previous step is used to determine the new solution. If the computed solutions at the earlier steps are used to determine the new solution, the method is called the multistep method. The Adams-Basforth and Adams-Moulton methods are the multistep method that can yield more accurate solution than the solution obtained from the one-step method.

Concepts and theoretical formulation of the one-step and multistep methods were presented in details in this chapter. Examples were presented so that detailed computational procedure can be clearly understood. Corresponding computer programs were also developed and their usage were demonstrated. These computer programs can be modified to solve other differential equations encountered in research and applications.

Exercises

1. Use the Euler method to solve the ordinary differential equation

$$\frac{dy}{dx} = 1 + \frac{y}{x}$$

for $1 \leq x \leq 2$ with the initial condition of $y(1)=2$. Use the step size of $h = 0.25$ in the computation. Plot to compare the computed solution with the exact solution of $y(x) = x \ln x + 2x$.

2. Solve the ordinary differential equation in Problem 1 again but by developing a computer program. Use three different step sizes of $h = 0.25, 0.1$ and 0.01 in the computation. Compare the solutions obtained from using the three step sizes with the exact solution.

3. Use the Euler method to solve the ordinary differential equation

$$\frac{dy}{dx} = 1 + \frac{y}{x} + \left(\frac{y}{x}\right)^2$$

for $1 \leq x \leq 3$ with the initial condition of $y(1) = 0$. Use the step size of $h = 0.2$ in the computation. Plot to compare the computed solution with the exact solution of $y(x) = x \tan(\ln x)$.

4. Solve the ordinary differential equation in Problem 3 again but by developing a computer program. Use three different step sizes of $h = 0.2, 0.1$ and 0.01 in the computation. Compare the solutions obtained from using the three step sizes with the exact solution.
5. Use the Euler method to solve the ordinary differential equation

$$\frac{dy}{dx} = -y + x + 1$$

for $0 \leq x \leq 5$ with the initial condition of $y(0) = 1$ by developing a computer program. Employ three different step sizes of $h = 0.5, 0.1, 0.01$ in the computation. Compare the computed solutions obtained from using the three step sizes with the exact solution of $y(x) = e^{-x} + x$.

6. Use the Heun's and the modified Euler methods to solve the ordinary differential equation

$$\frac{dy}{dx} = 4x - \frac{2y}{x}$$

for $1 \leq x \leq 2$ with the initial condition of $y(1) = 1$. Use the step size of $h = 0.25$ in the computation. Show detailed computations and plot to compare the solution obtained from each case with the exact solution of $y(x) = x^2$.

7. Solve the ordinary differential equation in Problem 6 again but by developing a computer program. Use three different step sizes of $h = 0.25, 0.1$ and 0.01 in the computation. In each case, show the comparison between the computed and exact solutions together with the true error.
8. Show that the solution error produced by using the Euler method is of order h , i.e., $O(h)$. Then, show such solution error by using an example with the step sizes of $h, h/2$ and $h/4$. Plot the solution error versus the step size h on the logarithmic scale.
9. Show that the solution error produced by using the Heun's method is second-order of h , i.e., $O(h^2)$. Then, show such solution error by using an example with the step sizes of $h, h/2$ and $h/4$. Plot the solution error versus the step size h on the logarithmic scale.
10. Use the Heun's and the modified Euler methods to solve the ordinary differential equation

$$\frac{dy}{dx} - \cos 2x - \sin 3x = 0$$

for $0 \leq x \leq 1$ with the initial condition of $y(0)=1$. Use the step size of $h = 0.25$ in the computation. Show detailed computational procedures. Plot to compare the solutions obtained from the two methods with the exact solution of

$$y(x) = \frac{1}{2} \sin 2x - \frac{1}{3} \cos 3x + \frac{4}{3}$$

11. Solve the ordinary differential equation in Problem 10 again by developing a computer program. Then, use the step sizes of $h = 0.25, 0.1$ and 0.01 in the computation. Set up a table to compare the computed solution at different steps with the exact solution.
12. Use the Heun's and the modified Euler methods to solve the ordinary differential equation

$$\frac{dy}{dx} - (1-y)(4-y) = 0$$

for $0 \leq x \leq 1$ with the initial condition of $y(0)=0$. Use the step size of $h = 0.25$ in the computation. Show detailed computational procedures. Plot to compare the solutions obtained from the two methods with the exact solution of

$$y(x) = \frac{4(e^{3x} - 1)}{(4e^{3x} - 1)}$$

13. Solve the ordinary differential equation in Problem 12 again by developing a computer program. Then, obtain the solutions by using the step sizes of $h = 0.25, 0.1$ and 0.01 . Plot to compare the solutions at different steps with the exact solution. Also, set up a table to show the true errors produced by the two methods.
14. Use the Heun's and the modified Euler methods to solve the ordinary differential equation

$$\frac{dy}{dx} + 2xy^2 = 0$$

for $0 \leq x \leq 20$ with the initial condition of $y(0)=1$. Develop a computer program to solve for the solutions by using the step size of $h = 0.1$. Compare the solutions at every $\Delta x = 1$ with the exact solution of $y(x) = 1/(1+x^2)$.

15. Use the second-, third- and fourth-order Runge-Kutta methods to solve the ordinary differential equation

$$\frac{dy}{dx} = -2xy^2$$

for $1 \leq x \leq 2$ with the initial condition of $y(1)=1$. Employ the step size of $h = 0.25$ in the computation. Show detailed computational procedures. Set up a table to compare the solutions obtained from the three methods with the exact solution of $y(x) = 1/x^2$.

16. Repeat Problem 12 again but by developing computer programs. Determine the solutions by using the step sizes of $h = 0.25, 0.1$ and 0.01 . Plot to compare the solutions obtained from using different step sizes with the exact solution. Also, set up a table to show the true errors produced by the three solutions from each method.
17. Use the fourth-order Runge-Kutta method to solve the ordinary differential equation

$$\frac{dy}{dx} - y = x^2$$

for $1 \leq x \leq 2$ with the initial condition of $y(1) = 1$ by developing a computer program. Then, use the step sizes of $h = 0.1, 0.01, 0.001$ and 0.0001 to obtain solutions. Set up a table to compare the solutions obtained with the exact solution of $y(x) = 6e^{x-1} - x^2 - 2x - 2$.

18. Use the fourth-order Runge-Kutta method to solve the ordinary differential equation

$$\frac{dy}{dx} + y^2 + \frac{y}{x} = \frac{1}{x^2}$$

for $1 \leq x \leq 2$ with the initial condition of $y(1) = -1$. Employ the step size of $h = 0.25$ in the computation. Plot to compare the solution with the exact solution of $y(x) = -1/x$.

19. Solve the ordinary differential equation in Problem 18 again by using the computer programs developed for the Euler, Heun's, modified Euler and fourth-order Runge-Kutta methods. Use the step size of $h = 0.05$ in the computation for all cases. Set up a table to show the computed solutions and the exact solution with the true errors. Provide comments on the solution accuracy and computational time for each method.
20. Use the second-, third- and fourth-order Runge-Kutta methods to solve the ordinary differential equation

$$\frac{dy}{dx} - \frac{2}{x}y = x^2 e^x$$

for $1 \leq x \leq 2$ with the initial condition of $y(1) = 0$ by using computer programs. Employ the step size of $h = 0.1$ in the computation for all cases. Set up a table to compare the solutions with the exact solution of $y(x) = x^2(e^x - e)$.

21. Employ the Euler and the fourth-order Runge-Kutta methods to solve the ordinary differential equation

$$\frac{dy}{dx} = \frac{e^x}{y}$$

for $0 \leq x \leq 2$ with the initial condition of $y(0) = 1$. Use the step size of $h = 0.5$ in the computation. Show detailed computational procedures. Plot to compare the solutions obtained from the two methods with the exact solution of $y(x) = \sqrt{2e^x - 1}$.

22. Use the Euler and the fourth-order Runge-Kutta methods to solve the ordinary differential equation

$$x \frac{dy}{dx} - 4y = x^5 e^x$$

for $1 \leq x \leq 2$ with the initial condition of $y(1) = 0$. Select an appropriate step size h to obtain the solutions. Set up a table to compare the solutions obtained from using the two methods with the exact solution of $y(x) = x^4 (e^x - e)$.

23. Employ the MATLAB commands `ode23` and `ode45` to solve the ordinary differential equation

$$\frac{dy}{dt} = -(1 + x + x^2) - (2x + 1)y - y^2$$

for $0 \leq x \leq 3$ with the initial condition of $y(0) = -1/2$. Then, further use MATLAB to plot the solutions for comparing with the exact solution of $y(x) = -x - (e^x + 1)^{-1}$.

24. Use the MATLAB commands `ode23` and `ode45` to solve the ordinary differential equation

$$\frac{dy}{dx} = 3x - \frac{y}{x}$$

for $1 \leq x \leq 6$ with the initial condition of $y(1) = 0$. Also, use MATLAB to plot the solutions for comparing with the exact solution of $y(x) = x^2 - x^{-1}$. Repeat the problem again but by using the initial condition of $y(1) = 1$ for which the exact solution is $y(x) = x^2$.

25. Employ the MATLAB commands `ode23` and `ode45` to solve the ordinary differential equation

$$\frac{dy}{dx} = 7x^2 - \frac{4y}{x}$$

for $1 \leq x \leq 6$ with the initial condition of $y(1) = 2$. Then, further use MATLAB to plot the solutions for comparing with the exact solution of $y(x) = x^3 + x^{-4}$. Repeat the problem again but by using the initial condition of $y(1) = 1$ for which the exact solution is $y(x) = x^3$.

26. Solve the second-order ordinary differential equation

$$\frac{d^2y}{dx^2} - 2 \frac{dy}{dx} + y = x(e^x - 1)$$

for $0 \leq x \leq 1$ with the initial conditions of $y(0) = y'(0) = 1$ by using the fourth-order Runge-Kutta method. Use the step size of $h = 0.25$ in the computation. Compare the solution with the exact solution of

$$y(x) = e^x \left(\frac{x^3}{6} - x + 3 \right) - x - 2$$

27. Solve the second-order ordinary differential equation

$$x^2 \frac{d^2y}{dx^2} - 2x \frac{dy}{dx} + 2y = x^3 \ln x$$

for $1 \leq x \leq 2$ with the initial conditions of $y(1) = 1$ and $y'(1) = 0$ by developing the computer programs for the Euler and the fourth-order Runge-Kutta methods. Then, use the programs to determine the solutions by using the step sizes of $h = 0.1$ and 0.01 . Compare the solution accuracy with the exact solution of

$$y(x) = \frac{7}{4}x + \frac{1}{2}x^3 \ln x - \frac{3}{4}x^3$$

by determining the true errors.

28. Solve the third-order ordinary differential equation

$$\frac{d^3y}{dx^3} + 6y^4 = 0$$

for $1 \leq x \leq 1.5$ with the initial conditions of $y(1) = -1$, $y'(1) = -1$, and $y''(1) = 2$ by developing the computer programs for the Euler and the fourth-order Runge-Kutta methods. Then, use the programs to determine the solutions by using the step sizes of $h = 0.1$ and 0.05 . Compare the solutions with the exact solution of $y(x) = (x - 2)^{-1}$ by plotting and determining the true errors.

29. Solve a set of nonlinear ordinary differential equations

$$\frac{dy_1}{dx} = y_1 - y_2 - y_1 y_3$$

$$\frac{dy_2}{dx} = y_1 + y_2 - y_2 y_3$$

$$\frac{dy_3}{dx} = y_1^2 + y_2^2 - y_3$$

for $0 \leq x \leq 10$ with the initial conditions of $y_1(0) = 2$, $y_2(0) = 0$ and $y_3(0) = 1$ by developing the computer programs for the Euler and the fourth-order Runge-Kutta methods. Then, employ the programs with appropriate step size h to determine their solutions. Give comments on how to measure the accuracy of the obtained solutions.

30. Show detailed derivation of the coefficients in Table 7.7 for the Adams-Bashforth formula.

31. Solve the ordinary differential equation in Example 7.12 again but by using a computer program developed for the sixth-order Adams-Bashforth formula. Compare the computed solution with the solutions obtained from the fourth-order Adams-Bashforth and the fourth-order Runge-Kutta methods. Provide comments on the solution accuracy obtained these methods.

32. Show detailed derivation of the coefficients in Table 7.8 for the Adams-Moulton formula.
33. Develop a computer program for the fourth-order Adams-Moulton formula. Then, use the program to solve the ordinary differential equation in Example 7.13 and compare the solution obtained with that shown in Table 7.9.
34. Solve the ordinary differential equation in Example 7.13 again but by using a computer program developed for the sixth-order Adams-Moulton formula. Compare the computed solution with the solutions from the fourth-order Adam-Bashforth and the fourth-order Adams-Moulton formulas as shown in Table 7.9.
35. Solve the ordinary differential equation in Problem 19 again but by using the non-self-starting Heun's method and the fourth-order Adams-Bashforth method. Compare the solutions obtained from the two methods with that from the fourth-order Runge-Kutta method. Give comments on the solution accuracy of these methods.
36. Solve the ordinary differential equation in Problem 17 again but by developing the computer programs for the fourth-order Adams-Bashforth and Adams-Moulton methods. Use the step size of $h = 0.05$ for obtaining the solutions. Compare the solutions with that from the fourth-order Runge-Kutta method.
37. A lumped mass of $m = 12 \text{ kg}$ with the initial temperature of $T_0 = 100^\circ\text{C}$ is dropped into a water reservoir at the temperature of $T_a = 30^\circ\text{C}$. The transient temperature response T of the lumped mass that varies with time t is determined from the ordinary differential equation

$$\frac{dT}{dt} + \frac{hA}{mc}(T - T_a) = 0$$

Where $h = 425 \text{ J/sec-m}^2\text{-}^\circ\text{C}$ is the convection coefficient of the lumped mass surface, $A = 1 \text{ m}^2$ is the lumped mass surface area, $c = 930 \text{ J/kg}\cdot{}^\circ\text{C}$ is the lumped mass specific heat. Employ the Euler and the fourth-order Runge-Kutta methods to determine the transient temperature response $T(t)$ of the lumped mass from 0 to 20 sec by selecting an appropriate time step. Set up a table to compare the solutions with the exact solution.

38. A lumped mass of $m = 35 \text{ kg}$ with the initial temperature of $T_0 = 2000 \text{ K}$ radiates heat from its surface of $A = 1.5 \text{ m}^2$ to a medium at the temperature of $T_a = 100 \text{ K}$. The lumped transient temperature response T that varies with time t is determined from the ordinary differential equation

$$\frac{dT}{dt} + \frac{\varepsilon\sigma A}{mc}(T^4 - T_a^4) = 0$$

where $\varepsilon = 0.7$ is the surface emissivity, $\sigma = 5.67 \times 10^{-8} \text{ J/sec-m}^2\text{-K}^4$ is the Stefan-Boltzmann constant, and $c = 445 \text{ J/kg}\cdot\text{K}$ is the specific heat of the lumped mass. Employ the Euler and the fourth-order Runge-Kutta methods to determine the transient temperature response $T(t)$ of the lumped mass from 0 to 10 sec. Select an appropriate time step to obtain solutions from the two methods. Provide comments on how to measure accuracy of the solutions obtained from the two methods.

39. If the surface convection coefficient of the lumped mass varies linearly with the temperature, the governing equation for determining the transient temperature response T that varies with time t is in a form of nonlinear ordinary differential equation

$$\frac{dT}{dt} + (a + bT)T = 0$$

For the initial lumped mass temperature of $T_0 = 100$ where $a = 1$ and $b = 0.03$, employ the Euler and the fourth-order Runge-Kutta methods to determine the lumped mass temperature response $T(t)$ for $0 \leq t \leq 10$. Use the time steps of $h = 0.2, 0.1$ and 0.05 in the computation. Plot to compare the solutions obtained from both methods with the exact solution of

$$T(t) = aT_0 e^{-at} / [a + bT_0(1 - e^{-at})]$$

40. Solve Problems 39 again by using a computer program developed for the fourth-order Adams-Basforth formula. Compare the solution with the exact solution and the solution obtained from the fourth-order Runge-Kutta method.
41. Solve Problem 39 again by using a computer program developed for the fourth-order Adams-Moulton formula. Explain detailed computational procedures used in the program. Compare the solution obtained from the program with the exact solution.

Chapter

8

Partial Differential Equations

8.1 Introduction

Most of scientific and engineering problems are governed by partial differential equations that describe their physical phenomena. For example, in the analysis of deformation and stresses in a plate subjected to an in-plane loading, the governing differential equations representing the equilibrium of forces must be solved. Similarly, in the analysis of temperature distribution in a plate under a specified heating on its surface, the governing differential equation representing the conservation of energy at any point on the plate must also be solved. The differential equations that occur in scientific and engineering problems are in different forms. Thus, different types of numerical methods are needed to obtain accurate solutions. This chapter begins with the classifications of partial differential equations. Appropriate numerical methods for solving the different classes of partial differential equations will then be explained. Several examples will be presented to aid understanding of the methods as well as physical meanings of the problems and their solutions. Such understanding will help solving more difficult problems that are governed by complex partial differential equations.

8.1.1 Definitions

Differential equation is called a partial differential equation if its dependent variable varies with two or more independent variables. For example,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad (8.1)$$

where u is the dependent variable that varies with the independent variables x and y . Equation (8.1) is called a second-order differential equations because the highest partial derivative of u is two. It is noted that most of the differential equations that occur in scientific and engineering problems contain partial

derivatives of the dependent variables up to the order of four. The independent variables are the three spatial coordinates x, y, z for general three-dimensional problems and the time t .

A partial differential equation is *linear* if the coefficients of all terms are constant or function of the independent variables only equations (8.2) and (8.3) below are linear differential equations because the coefficient of each term is either constant or function of x and y .

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 7u = 12 \quad (8.2)$$

and $(x^2 - 3y) \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y} + \frac{\partial u}{\partial y} = y^3 \quad (8.3)$

But the partial differential equation,

$$\frac{\partial^2 u}{\partial x^2} + \left(\frac{\partial u}{\partial y} \right)^{0.5} = 0 \quad (8.4)$$

is *nonlinear* because the second term has its power order of a half, i.e., not an integer. Also, the partial differential Eq. (8.5) below is nonlinear,

$$u \frac{\partial u}{\partial x} + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial u}{\partial x} \right) \left(\frac{\partial u}{\partial y} \right) = u^2 \quad (8.5)$$

because the coefficients of all terms are function of the dependent variable u . Nonlinear partial differential equations always occur in practical problems and require complex numerical methods for solving them.

Some simple linear partial differential equations can be solved for exact solutions by using advanced mathematical techniques. However, they cannot be solved if the shapes of the problems are irregular. Thus, numerical methods are needed to obtain approximate solutions instead. The popular numerical methods are the finite difference method, the finite element method and the finite volume method. The finite difference method is considered as the simplest one. The method is easy to understand and convenient to apply to problems with regular shapes. The finite difference method will be explained in details for solving different types of the partial differential equations in this chapter. For the problems with irregular shapes, the finite element method is more efficient and widely used. Details of the finite element method are presented in the next chapter.

8.1.2 Types of equations

Because the partial differential equations that arise in scientific and engineering problems are in many forms, the popular forms are considered herein. These popular forms can be written in general as,

$$a \frac{\partial^2 u}{\partial x^2} + b \frac{\partial^2 u}{\partial x \partial y} + c \frac{\partial^2 u}{\partial y^2} = f \quad (8.6)$$

where a, b, c may be constants or function of x and y . The function f on the right-hand side of the above equation may be constant or function of $x, y, u, \partial u / \partial x$ and $\partial u / \partial y$. Equation (8.6) can be classified into different types as follows.

- (a) Equation (8.6) is called the *Elliptic equation* if $b^2 - 4ac < 0$. The Laplace's equation which has the form of

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad (8.7)$$

is an example of the elliptic equation. In general, the solutions of the elliptic equation are smooth. Steady-state heat transfer in a plate is an example problem that is governed by such differential equation. The dependent variable u in Eq. (8.7) represents the temperature distribution that varies with x - y coordinates of the plate as shown in Fig. 8.1. The finite difference method for solving the elliptic equation will be presented in section 8.2.

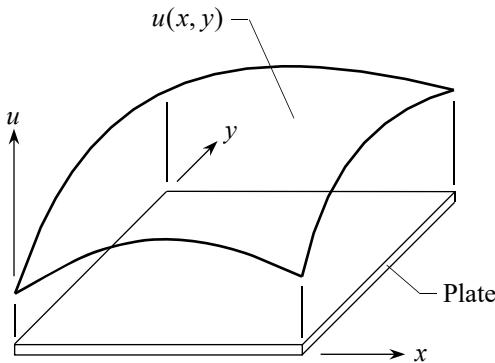


Figure 8.1 Temperature distribution on a plate governed by the elliptic partial differential equation.

(b) The partial differential Eq. (8.6) is called the *Parabolic equation* if $b^2 - 4ac = 0$. An example problem that is governed by the parabolic equation is the transient heat conduction in a bar. The governing partial differential equation is,

$$k \frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \quad (8.8)$$

where k is the thermal conductivity coefficient of the bar material. The transient temperature u varies with x -coordinate of the bar and time t . Typical transient temperature distributions along the bar at different times are shown in Fig. 8.2. Details of the finite difference method for solving such parabolic equation will be explained in section 8.3.

(c) The partial differential Eq. (8.6) is called the *Hyperbolic equation* if $b^2 - 4ac > 0$. An example problem is the oscillation behavior of a string that is governed by,

$$k^2 \frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2} \quad (8.9)$$

where k^2 represents tension in the string which is always positive. The deflection u varies with x -coordinate of the string and time t . Typical string deflection behaviors are shown in Fig. 8.3. The finite difference method for solving such hyperbolic equation will be presented in section 8.4.

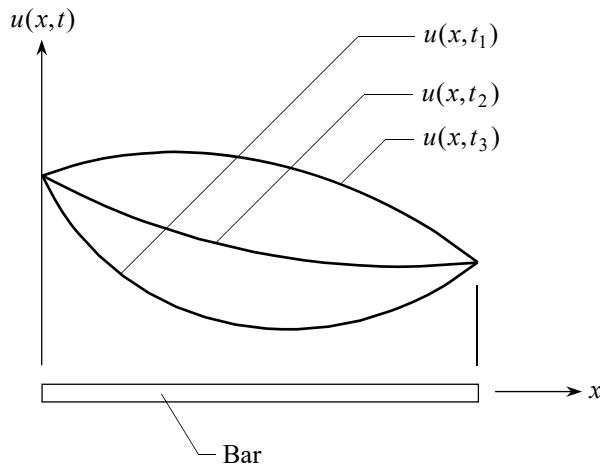


Figure 8.2 Transient temperature distributions in a bar which is governed by the parabolic differential equation.

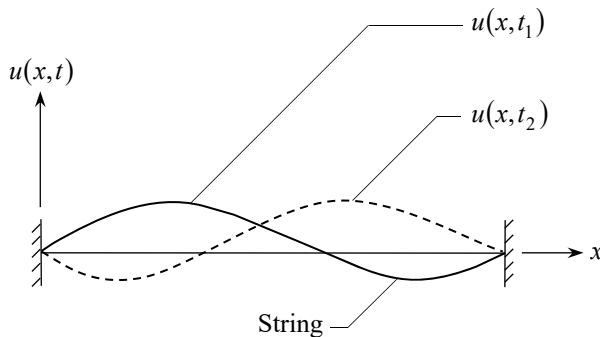


Figure 8.3 Behavior of string oscillation which is governed by the hyperbolic differential equation.

8.1.3 Boundary and initial conditions

Solution behaviors obtained from solving the partial differential Eqs. (8.7), (8.8) and (8.9) which are in the form of the elliptic, parabolic and hyperbolic equations, respectively, depend on the boundary and initial conditions. Two types of the boundary conditions frequently encountered are:

(a) *Dirichlet condition.* The condition specifies values of the dependent variable u at the boundaries. For example, temperature values are specified at both ends of the bar in Fig. 8.2.

(b) *Neumann condition.* The condition specifies values of the dependent variable gradient $\partial u / \partial x$. For example, if one end of the bar in Fig. 8.2 is insulated, then the slope of temperature $\partial u / \partial y = 0$ at that end.

The initial condition is used at the beginning of the solution process. For example, the temperature distribution along the bar in Fig. 8.2 may be specified by a function $u(x, 0) = f(x)$ at time $t = 0$.

Appropriate boundary and initial conditions are applied in the solving process of the elliptic, parabolic and hyperbolic equations as will be demonstrated in the following sections.

8.2 Elliptic Equation

8.2.1 Differential equation

An example of steady-state heat conduction in a plate is used herein to aid understanding for solving the elliptic equation by using the finite difference method. The problem is selected because heat transfer phenomenon and the temperature solution are easy to understand. Figure 8.4 shows a rectangular plate that lies in x - y coordinates under the steady-state conduction heat transfer. The figure also shows an infinitesimal element with the sizes of Δx and Δy . The plate has the thickness of t and is made from a material with the thermal conductivity coefficient of k .

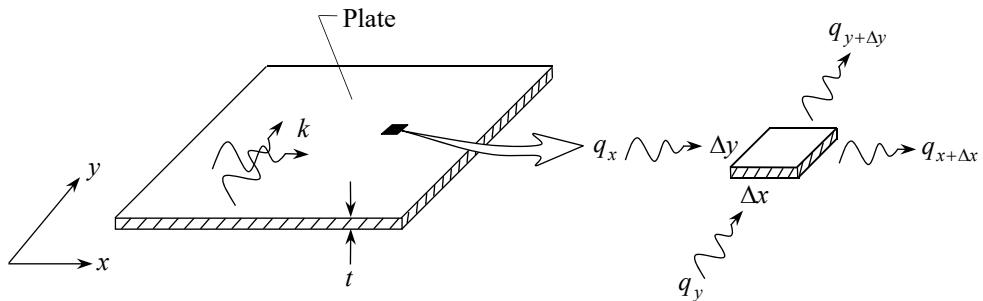


Figure 8.4 Conduction heat transfer in a plate.

To derive the governing differential equation, the conservation of energy is applied to the infinitesimal element such that,

$$\text{Heat flux in} - \text{Heat flux out} = 0 \quad (8.10)$$

$$\text{i.e., } (q_x + q_y) - (q_{x+\Delta x} + q_{y+\Delta y}) = 0 \quad (8.11)$$

where q_x and q_y are the heat fluxes in x - and y -directions, respectively. These heat fluxes depend on the temperature gradient according to the Fourier's law,

$$q_x = -k(t\Delta y) \frac{\partial T}{\partial x} \quad (8.12a)$$

$$q_y = -k(t\Delta x) \frac{\partial T}{\partial y} \quad (8.12b)$$

By substituting Eqs. (8.12a-b) into Eqs. (8.11) and applying the Taylor series expansion in the same fashion as shown in Eq. (2.29) onto $q_{x+\Delta x}$ and $q_{y+\Delta y}$,

$$\begin{aligned}
& -k(t\Delta y)\frac{\partial T}{\partial x} - k(t\Delta x)\frac{\partial T}{\partial y} \\
& + k(t\Delta y)\frac{\partial T}{\partial x} + \frac{\partial}{\partial x}\left[k(t\Delta y)\frac{\partial T}{\partial x}\right]\Delta x + \frac{1}{2}\frac{\partial^2}{\partial x^2}\left[k(t\Delta y)\frac{\partial T}{\partial x}\right](\Delta x)^2 + \dots \\
& + k(t\Delta x)\frac{\partial T}{\partial y} + \frac{\partial}{\partial y}\left[k(t\Delta x)\frac{\partial T}{\partial y}\right]\Delta y + \frac{1}{2}\frac{\partial^2}{\partial y^2}\left[k(t\Delta x)\frac{\partial T}{\partial y}\right](\Delta y)^2 + \dots = 0
\end{aligned}$$

Or,

$$\begin{aligned}
& \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)t\Delta x\Delta y + \frac{1}{2}\frac{\partial^2}{\partial x^2}\left(k\frac{\partial T}{\partial x}\right)t(\Delta x)^2\Delta y + \dots \\
& + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right)t\Delta x\Delta y + \frac{1}{2}\frac{\partial^2}{\partial y^2}\left(k\frac{\partial T}{\partial y}\right)t\Delta x(\Delta y)^2 + \dots = 0
\end{aligned} \tag{8.13}$$

Equation (8.13) is then divided through by $t\Delta x\Delta y$. By letting $\Delta x \rightarrow 0$ and $\Delta y \rightarrow 0$, Eq. (8.13) becomes,

$$\frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right) + \frac{\partial}{\partial y}\left(k\frac{\partial T}{\partial y}\right) = 0 \tag{8.14}$$

If the thermal conductivity coefficient k is constant, then the partial differential Eq. (8.14) reduces to

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{8.15}$$

Equation (8.15) is in the form of the elliptic equation as shown in Eq. (8.7). This means the steady-state temperature distribution in a plate is determined by solving the elliptic equation which is in the form of the Laplace's equation.

If the plate is subjected to a specified heating on its surface, the corresponding partial differential equation can be derived in the same way. In this later case, the partial differential equation is in the form,

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = f(x, y) \tag{8.16}$$

where $f(x, y)$ denotes the specified heating function which may vary with x - and y -coordinates. The partial differential equation in the form of Eq. (8.16) is known as the Poisson's equation.

8.2.2 Computational procedures

To ease understanding on the application of the finite difference method for solving the elliptic equation, the problem of steady-state heat conduction in a rectangular plate is considered. The plate is first divided into a number of intervals in both x - and y -directions as shown in Fig. 8.5. The lengths of the intervals are Δx and Δy in x - and y -directions, respectively. The unknowns are at grid points where the horizontal and vertical lines intersect, e.g., at the location i, j , as shown in the figure.

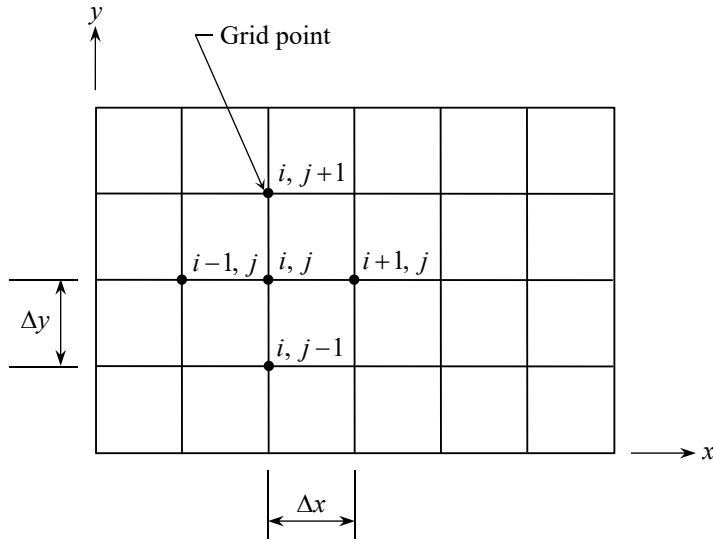


Figure 8.5 Dividing rectangular plate into intervals with unknowns at grid points by using the finite difference method.

The key step of the finite difference method is to transform the governing differential Eq. (8.15) into an algebraic equation. Herein, the central difference technique as shown in Table 6.6 is used to approximate the second-order derivative terms,

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} \quad (8.17a)$$

and

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} \quad (8.17b)$$

By substituting Eqs. (8.17a-b) into the Laplace's Eq. (8.15) and using $\Delta x = \Delta y$, the approximate differential equation is obtained,

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0 \quad (8.18)$$

Such expression can be written in a stencil form as,

$$\begin{array}{c}
 \textcircled{1} \\
 | \\
 \textcircled{1} -4 \textcircled{1} \textcircled{1} = 0 \\
 | \\
 \textcircled{1}
 \end{array} \quad (8.19)$$

The stencil form above is applied at the grid points where the temperatures are unknowns. For example, a rectangular plate with the size of 4×2 units as shown in Fig. 8.6 has specified temperatures along the four edges. If the plate is divided into 4 and 2 intervals in x - and y -directions, respectively, then the temperature unknowns are at the three grid points 2,2, 3,2 and 4,2.

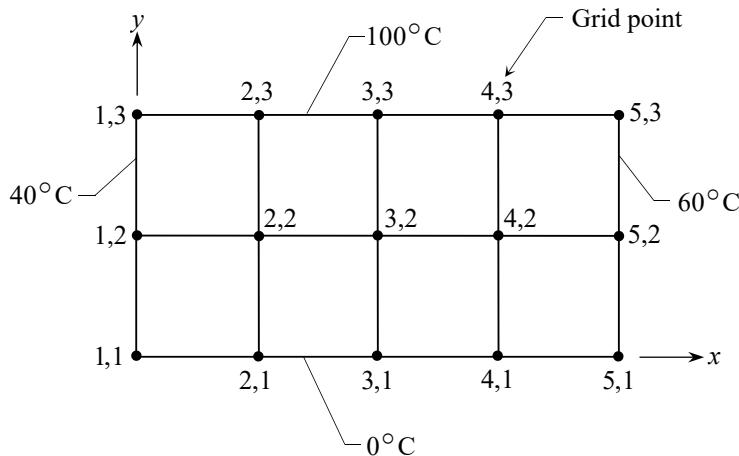


Figure 8.6 Application of the stencil form to establish a set of simultaneous equations for conduction heat transfer in a rectangular plate.

Application of the stencil form as shown in Eq. (8.19) at the grid point 2,2 gives the algebraic equation,

$$\begin{aligned} 100 + 40 + 0 + T_{3,2} - 4T_{2,2} &= 0 \\ 4T_{2,2} - T_{3,2} &= 140 \end{aligned} \quad (8.20)$$

Similarly, at the grid point 3,2

$$\begin{aligned} 100 + T_{2,2} + 0 + T_{4,2} - 4T_{3,2} &= 0 \\ -T_{2,2} + 4T_{3,2} - T_{4,2} &= 100 \end{aligned} \quad (8.21)$$

and at the grid point 4,2

$$\begin{aligned} 100 + T_{3,2} + 0 + 60 - 4T_{4,2} &= 0 \\ -T_{3,2} + 4T_{4,2} &= 160 \end{aligned} \quad (8.22)$$

The three Eqs. (8.20-8.22) lead to a set of simultaneous equations,

$$\begin{aligned} 4T_{2,2} - T_{3,2} &= 140 \\ -T_{2,2} + 4T_{3,2} - T_{4,2} &= 100 \\ -T_{3,2} + 4T_{4,2} &= 160 \end{aligned} \quad (8.23)$$

which can be written in matrix form,

$$\begin{bmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix} \begin{Bmatrix} T_{2,2} \\ T_{3,2} \\ T_{4,2} \end{Bmatrix} = \begin{Bmatrix} 140 \\ 100 \\ 160 \end{Bmatrix} \quad (8.24)$$

It should be noted that the square matrix on the left-hand side of Eq. (8.24) has positive coefficients along the diagonal line. The magnitudes of these coefficients are relatively large as compared to the other coefficients on the off-diagonal line. With such matrix property, the Gauss-Seidel iteration method is effective for solving the set of simultaneous equations. It is also noted that the procedure for solving such problem is sometimes called as the Liebmann's method. The method requires less computational time and memory as compared to the other direct methods, such as the Gauss elimination or the LU decomposition method, especially when the model contains a large number of grid points. The corresponding computer program can also be developed easily as will be shown in the following example.

8.2.3 Examples

In this section, an example for solving the elliptic equation by using the finite difference method is presented in details. A corresponding computer program is also developed to demonstrate the application of the method when the finite difference model contains many unknowns. The example has the exact solution of the temperature distribution, so that the finite difference solutions at grid points can be compared to measure the method efficiency.

Example 8.1 A rectangular plate with the size of 2×1 units has specified zero temperature along the left, lower and right edges as shown in Fig. 8.7. The plate is subjected to the specified temperature distribution that varies as a sine function along the upper edge as shown in the figure. Use the finite difference method to determine the plate temperature distribution by dividing the plate into 8 and 4 intervals in x - and y -directions, respectively.

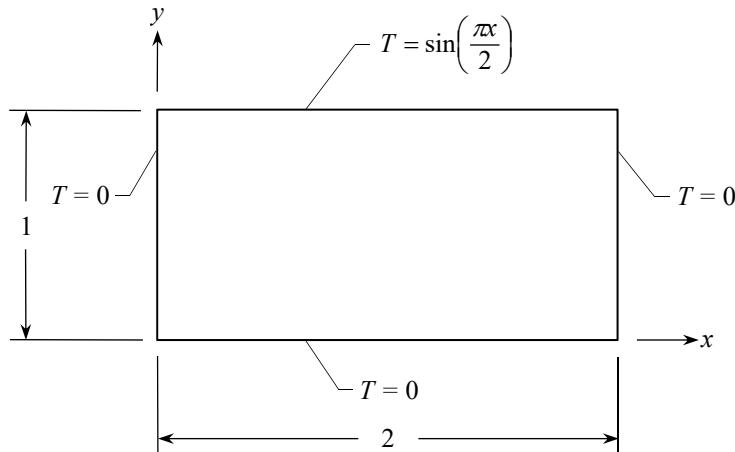


Figure 8.7 Plate with specified temperature along the four edges.

Compare the computed solutions with the exact temperature solution of,

$$T(x, y) = \sin \frac{\pi x}{2} \sinh \frac{\pi y}{2} / \sinh \frac{\pi}{2} \quad (8.25)$$

The plate temperature distribution according to the exact solution in Eq. (8.25) is shown as a carpet plot in Fig. 8.8. The temperature distribution is in the form of sine function along the upper edge and decays to zero along the left, lower and right edges.

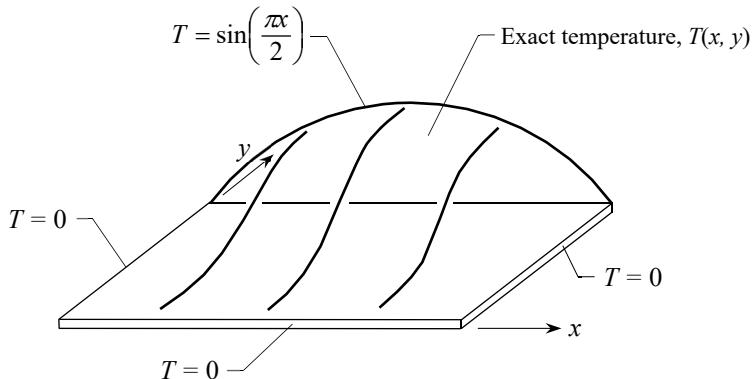


Figure 8.8 Exact temperature distribution over the plate according to Eq. (8.25).

The finite difference model with 8 and 4 intervals in x - and y -directions is shown in Fig. 8.9 with the grid point numbers. For this model, the lengths of the intervals are $\Delta x = \Delta y = 0.25$ unit. The model contains a total of 45 grid points for which 21 interior grid points are unknowns.

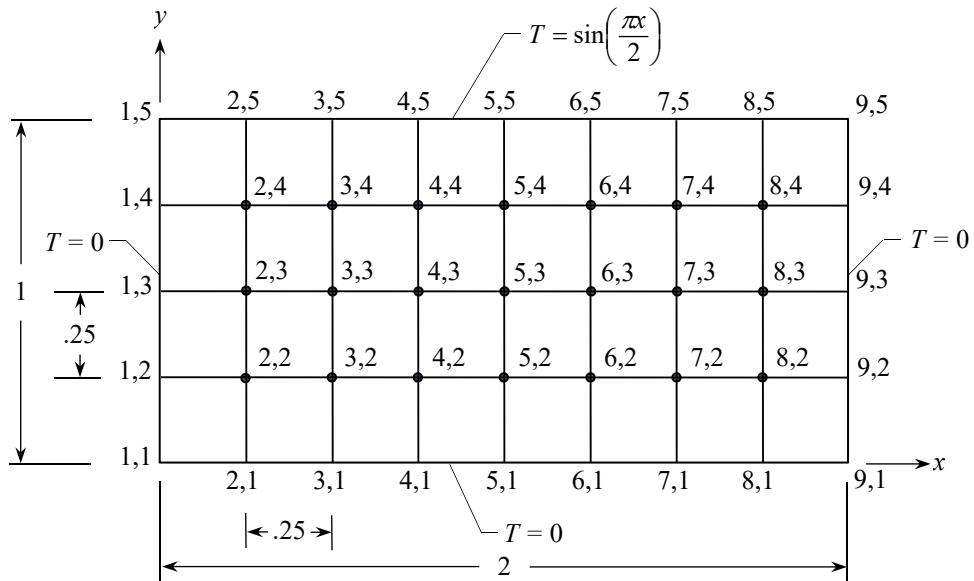


Figure 8.9 Finite difference model of the rectangular plate.

The stencil form as shown in Eq. (8.19) is applied to these 21 interior grid points together with the boundary conditions along the four edges as,

$$T(x, y = 0) = 0 \quad (8.26a)$$

$$T(x = 0, y) = 0 \quad (8.26b)$$

$$T(x = 2, y) = 0 \quad (8.26c)$$

and

$$T(x, y = 1) = \sin \frac{\pi x}{2} \quad (8.26d)$$

Such applications lead to a set of 21 algebraic equations. The algebraic equations are then solved by using the Gauss-Seidel iteration method for the temperature solutions at grid points from Eq. (8.18), i.e.,

$$T_{i,j} = (T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1}) / 4 \quad (8.27)$$

At the end of each iteration, the computed temperatures are compared with those obtained from the previous one. The iteration process is terminated if the temperature differences between the two successive iterations are less than the specified tolerance.

Because a typical finite difference model must contain a large number of grid points in order to obtain accurate solutions, a computer program should be developed for reducing the computational effort. Analysts can refine the model by using many small intervals in both x - and y -directions. For example, the plate can be divided into 200 and 100 intervals in x - and y -directions, respectively. In this latter case, the finite difference method will lead to a set of 19,701 equations which is impractical to solve them by hand.

Table 8.1 shows the solutions at grid points obtained from the finite difference method by using the computer program in Fig. 8.10. The computer program employs the Gauss-Seidel iteration technique to solve the set of simultaneous equations with the stopping tolerance criterion of 0.00001. The finite difference solutions are obtained after 24 iterations. These computed solutions at grid points are less than 1% difference from the exact solutions.

Table 8.1 Comparison of temperatures between the finite difference and exact solutions (values in bracket). Locations of numbers correspond to grid point locations in Fig. 8.9.

0.000	0.383	0.707	0.924	1.000	0.924	0.707	0.383	0.000
(0.000)	(0.383)	(0.707)	(0.924)	(1.000)	(0.924)	(0.707)	(0.383)	(0.000)
0.000	0.245	0.453	0.592	0.641	0.592	0.453	0.245	0.000
(0.000)	(0.244)	(0.452)	(0.590)	(0.639)	(0.590)	(0.452)	(0.244)	(0.000)
0.000	0.145	0.269	0.351	0.380	0.351	0.269	0.145	0.000
(0.000)	(0.144)	(0.267)	(0.349)	(0.377)	(0.349)	(0.267)	(0.144)	(0.000)
0.000	0.068	0.125	0.163	0.177	0.163	0.125	0.068	0.000
(0.000)	(0.067)	(0.124)	(0.162)	(0.175)	(0.162)	(0.124)	(0.067)	(0.000)
0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)	(0.000)

```
% Program Ellip1
% A Finite difference program for solving
% temperature distribution in a plate.
% Assign the tolerance and maximum number
% of iterations:
tol = 0.00001; mxiter = 100;
% Set up boundary conditions:
t = zeros(5,9); x = 0.25; dx = x;
for i = 2:8
    t(1,i) = sin(pi*x/2);
    x = x + dx;
end
% Set up initial temperature values:
for i = 2:4
    for j = 2:8
        t(i,j) = j*t(i,5)/5.;
    end
end
% Solve for unknown temperatures at the
% grid points by using the Gauss-Seidel
% iteration technique:
for iter = 1:mxiter
    iflag = 0;
    for i = 2:4
        for j = 2:8
            temp = (t(i-1,j)+t(i+1,j)+t(i,j+1)+ ...
                    t(i,j-1))/4;
            diff = t(i,j) - temp;
            if abs(diff) > tol
                iflag = 1;
            end
            t(i,j) = temp;
        end
        if iflag == 0.
            continue
        end
    end
    while iflag == 1
        fprintf(' Solution is not converged')
        fprintf(' within the specified number')
        fprintf(' of iterations and tolerance')
    end
    % Print out temperatures at grid points
    % in the format corresponding to the
    % problem figure:
    for i = 1:5
        fprintf('\n');
        for j = 1:9
            fprintf('%6.3f', t(i,j));
        end
    end
    % Surface plot of temperature distribution:
    xc = 0:dx:2; yc = 0:dx:1; tplot = flipud(t);
    [XC,YC] = meshgrid(xc,yc);
    mesh(XC,YC,tplot); colormap([0,0,0])
end
```

Figure 8.10 Computer program for solving grid point temperatures on a rectangular plate using the finite difference method.

Example 8.2 From the problem statement and the plate temperature distribution in example 8.1, it can be seen that the problem has solution symmetry. Thus, only one half of the plate can be used in the analysis to obtain the same temperature solution. Using only one half of the plate can reduce the total number of unknowns and thus the computational time. Figure 8.11 shows only the left half of the plate that can be used in the analysis. The boundary conditions are identical to those of the original plate except the condition on the right boundary which represents zero conduction heat transfer.

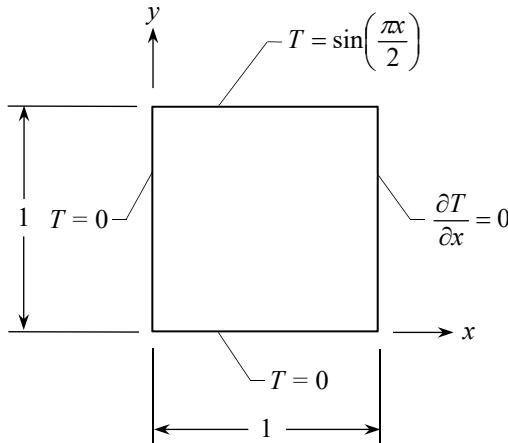


Figure 8.11 Determination of temperature distribution using only the left half of the plate due to the solution symmetry.

The computational procedures follow those explained in example 8.1. The left half of the plate is first divided into intervals in both x - and y -directions as shown in Fig. 8.12. The unknowns are at the interior grid points and the grid points along the right model boundary. Thus, there is a total of only 12 unknowns for the half model of the plate.

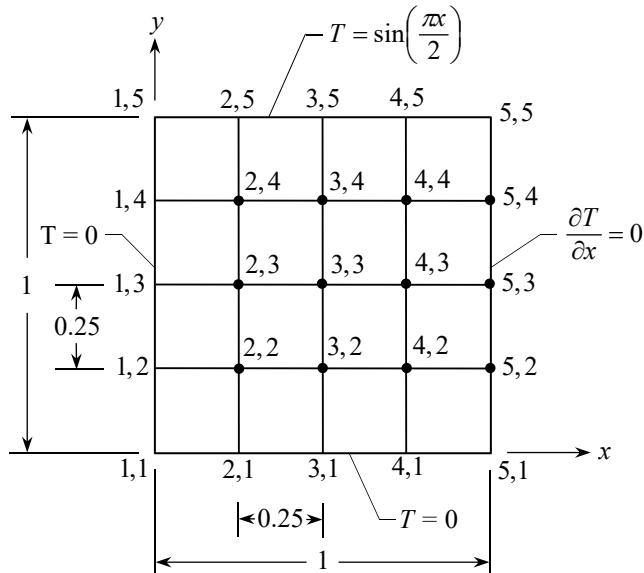


Figure 8.12 Finite difference model for only half of the plate.

The boundary conditions along the four edges of the half model in Fig. 8.12 are:

$$T(x, y=0) = 0 \quad (8.28a)$$

$$T(x=0, y) = 0 \quad (8.28b)$$

$$\frac{\partial T}{\partial x}(x=1, y) = 0 \quad (8.28c)$$

$$T(x, y=1) = \sin \frac{\pi x}{2} \quad (8.28d)$$

The stencil form as shown in Eq. (8.19) can be used to establish the algebraic equations for the interior grid points except those on the right boundary. A new stencil form must be developed for the grid points along the right boundary. Figure 8.13 shows the grid points along the right boundary with a fictitious grid point at $i+1, j$.

By using the central difference technique as shown in Table 6.6, the first-derivative of the temperature at grid point i, j in x -direction as shown in Fig. 8.13 is approximate by

$$\frac{\partial T}{\partial x} = \frac{T_{i+1, j} - T_{i-1, j}}{2 \Delta x} \quad (8.29)$$

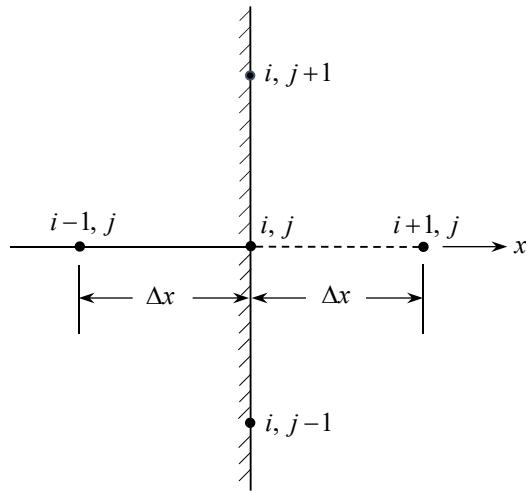


Figure 8.13 Derivation of stencil form along model boundary with a fictitious grid point.

Thus, the temperature at grid point $i+1, j$ is

$$T_{i+1, j} = 2 \Delta x \frac{\partial T}{\partial x} + T_{i-1, j} \quad (8.30)$$

By substituting Eq. (8.30) into Eq. (8.18), the temperature at the grid point along the right boundary can be determined from

$$2 \Delta x \frac{\partial T}{\partial x} + T_{i-1, j} + T_{i-1, j} + T_{i, j+1} + T_{i, j-1} - 4T_{i, j} = 0$$

$$T_{i,j} = \left(2\Delta x \frac{\partial T}{\partial x} + 2T_{i-1,j} + T_{i,j+1} + T_{i,j-1} \right) / 4 \quad (8.31)$$

It is noted that if the right boundary of the model is subjected to a specified heating, heat convection or radiation, the term $\partial T/\partial x$ in Eq. (8.31) must be replaced by an appropriate value according to the heat mode. But for this example, $\partial T/\partial x$ is zero along the right boundary, thus the equation for determining the temperature for a grid point along the right boundary reduces to

$$T_{i,j} = (2T_{i-1,j} + T_{i,j+1} + T_{i,j-1}) / 4 \quad (8.32)$$

The computer program as shown in Fig. 8.10 can be modified slightly to solve for the temperatures for only half of the plate. The stencil form in Eq. (8.19) is used for the interior grid points, while Eq. (8.32) is applied to the grid points along the right boundary. The computed temperatures at the grid points are shown in Table 8.2. These temperature solutions are identical to those obtained from using the full plate model as shown in Table 8.1. The later example 8.2 demonstrates that if a problem has solution symmetry, the finite difference model that reflects such the solution symmetry should be used. The symmetry model helps reducing both the number of unknowns and computational time.

Table 8.2 Comparison of temperatures between the finite difference and exact solutions (values in bracket) for half of the plate. Locations of numbers correspond to grid point locations in Fig. 8.12.

0.000 (0.000)	0.383 (0.383)	0.707 (0.707)	0.924 (0.924)	1.000 (1.000)
0.000 (0.000)	0.245 (0.244)	0.453 (0.452)	0.592 (0.590)	0.641 (0.639)
0.000 (0.000)	0.145 (0.144)	0.269 (0.267)	0.351 (0.349)	0.380 (0.377)
0.000 (0.000)	0.068 (0.067)	0.125 (0.124)	0.163 (0.162)	0.177 (0.175)
0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)	0.000 (0.000)

Example 8.3 Develop a computer program to solve the Poisson's equation in the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 2x(x^3 - 6xy + 6xy^2 - 1) \quad (8.33)$$

for the square domain of $0 \leq x \leq 1$ and $0 \leq y \leq 1$, with the boundary condition of $u(x, y) = 0$ along the four edges. Divide the domain into 20 equal intervals in both the x - and y -directions. Compare the obtained solution with the exact solution of

$$u(x, y) = (x - x^4)(y - y^2) \quad (8.34)$$

A short computer program **Ellip2** can be developed as shown in Fig. 8.14. User can increase the number of intervals to obtain a more accurate solution. Figure 8.15 compares the solution of $u(x, y)$ between the computed and exact solutions.

```
% Program Ellip2
% A Finite difference program for solving
% elliptic partial differential equation.
% Assign the number of intervals (equally
% in both x- and y-directions), tolerance
% and maximum number of iterations:
ne = 20; tol = 0.0001; mxiter = 1000;
np = ne + 1; dx = 1./ne; dy = dx;
u = zeros(np,np);
% Set up boundary conditions:
for i = 1:np
    u(1,i) = 0; u(np,i) = 0;
    u(i,1) = 0; u(i,np) = 0;
end
% Solve for unknowns at grid points using
% the Gauss-Seidel iteration technique:
for iter = 1:mxiter
    iflag = 0.; x = dx;
    for i = 2:ne
        y = dy;
        for j = 2:ne
            uu = (u(i-1,j)+u(i+1,j)+u(i,j+1)+ ...
                u(i,j-1))/4. - (x/2)*(x^3 - ...
                6*x*y + 6*x*y^2 - 1)*(dx^2);
            diff = u(i,j) - uu;
            if abs(diff) > tol
                iflag = 1;
            end
            u(i,j) = uu;
            y = y + dy;
        end
        x = x + dx;
    end
    if iflag == 0
        continue
    end
end
while iflag == 1
    fprintf(' Solution is not converged')
    fprintf(' within the specified number')
    fprintf(' of iterations and tolerance')
    break
end
% Surface plot of the exact solution:
xe = 0:dx:1; ye = 0:dy:1;
[XE,YE] = meshgrid(xe,ye);
ue = (XE-XE.^4).* (YE-YE.^2);
ueplot = flipud(ue);
mesh(XE,YE,ueplot); view(-30,30),
colormap([0,0,0]), figure
% Surface plot of the computed solution:
mesh(XE,YE,u); view(-30,30),
colormap([0,0,0])
```

Figure 8.14 A computer program for solving and plotting result of Example 8.3.

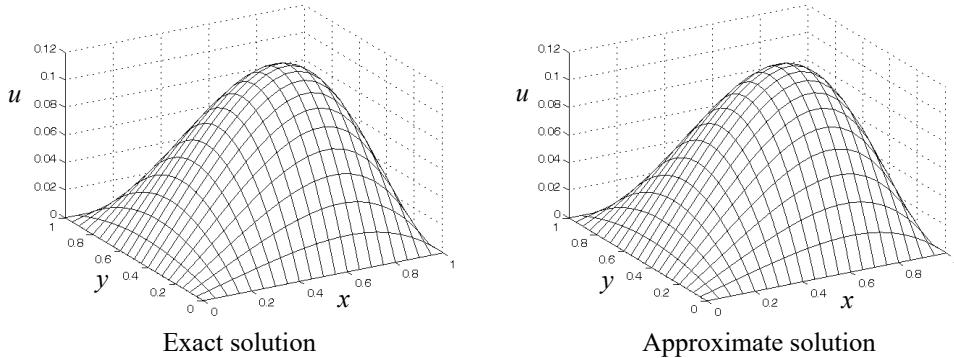


Figure 8.15 Comparison of the exact and computed solutions for Example 8.3.

8.3 Parabolic Equation

8.3.1 Differential equation

Parabolic equation is classified as a type of the partial differential equations that occurs in many scientific and engineering applications. One of the simplest examples used for studying the equation is the transient heat conduction in a bar as shown in Fig. 8.16. The bar with length L in x -coordinate direction is made from a material that has the thermal conductivity coefficient k , the mass density ρ and the specific heat c . The temperature varies with x -coordinate along the bar and time t . The partial differential equation in the form of parabolic equation can be derived from the conservation of energy by using a small element of length Δx ,

$$\text{Energy in} - \text{Energy out} = \text{Energy stored} \quad (8.35)$$

i.e.,

$$q_x - q_{x+\Delta x} = \rho c A \Delta x \frac{\partial T}{\partial t} \quad (8.36)$$

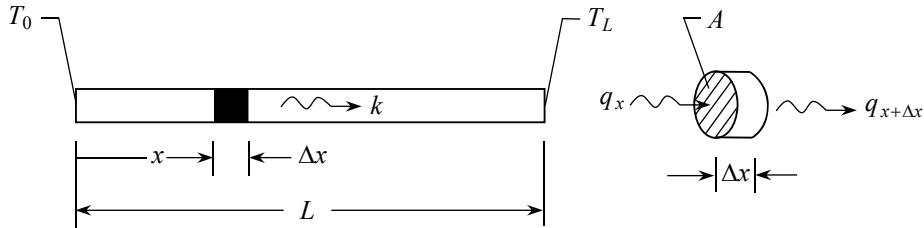


Figure 8.16 Transient heat conduction in a bar.

where q_x and $q_{x+\Delta x}$ are the heat fluxes going into and out from the cross-sectional area A of the small element as shown in the figure, respectively. The amount of heat flux depends on the temperature gradient from the Fourier's law,

$$q_x = -k A \frac{\partial T}{\partial x} \quad (8.37)$$

By substituting Eq. (8.37) into and Eq. (8.36) and applying the Taylor series expansion to the heat flux term $q_{x+\Delta x}$, Eq. (8.34) becomes,

$$-k A \frac{\partial T}{\partial x} - \left[-k A \frac{\partial T}{\partial x} - \frac{\partial}{\partial x} \left(k A \frac{\partial T}{\partial x} \right) \Delta x - \frac{1}{2} \frac{\partial^2}{\partial x^2} \left(k A \frac{\partial T}{\partial x} \right) (\Delta x)^2 - \dots \right] = \rho c A \Delta x \frac{\partial T}{\partial t}$$

After the first and second terms are cancelled, the equation is divided through by $A \Delta x$. Then, by letting $\Delta x \rightarrow 0$, the equation reduces to

$$\frac{\partial}{\partial x} \left(k \frac{\partial T}{\partial x} \right) = \rho c \frac{\partial T}{\partial t} \quad (8.38)$$

If the thermal conductivity coefficient k is constant, Eq. (8.36) leads to a parabolic partial differential equation representing the transient heat conduction in a bar as,

$$\frac{k}{\rho c} \frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t} \quad (8.39)$$

Different methods for solving the parabolic equation in the form of Eq. (8.39) are presented in the following sections. Advantages and disadvantages of each method are explained. Examples and corresponding computer programs are also presented to demonstrate the efficiency of each method. Understanding details of these methods can help analyzing large practical problems more effectively.

8.3.2 Explicit method

The explicit method is considered as the simplest method for solving the parabolic equation. The method starts from dividing the bar into equal intervals, each has the length of Δx , as shown in Fig. 8.17.

These intervals are connected at grid points $i-1, i, i+1$ for which the temperatures are unknown and to be determined.

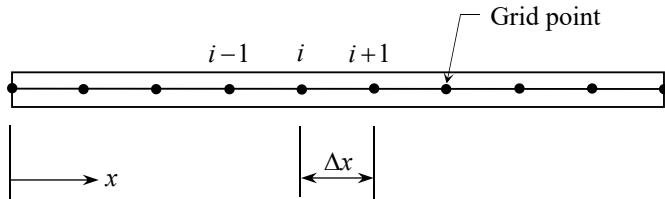


Figure 8.17 A bar is divided into intervals connected at grid points.

The temperatures T that vary with time t at grid points are determined from the parabolic differential Eq. (8.37). The forward difference approximation is applied to the first-order time derivative term as,

$$\frac{\partial T}{\partial t} = \frac{T_i^{n+1} - T_i^n}{\Delta t} \quad (8.40)$$

Such approximation has the error of order $O(\Delta t)$ where Δt is the time step. The superscript n denotes the n^{th} time step in the computation. Similarly, the central difference approximation can be applied to the second-order spatial derivative term,

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} \quad (8.41)$$

for which the error is of order $O(\Delta x^2)$. It is noted that the grid point temperatures in Eq. (8.41) are the values at the n^{th} time step. By substituting Eqs. (8.40) and (8.41) into the differential Eq. (8.39), an algebraic equation representing the partial differential equation is obtained in the form,

$$\frac{k}{\rho c} \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} = \frac{T_i^{n+1} - T_i^n}{\Delta t} \quad (8.42)$$

The terms in Eq. (8.42) are rearranged such that the grid point temperatures at time $n+1$ can be determined directly, i.e.,

$$T_i^{n+1} = T_i^n + \alpha(T_{i+1}^n - 2T_i^n + T_{i-1}^n) \quad (8.43)$$

where

$$\alpha = \frac{k \Delta t}{\rho c (\Delta x)^2} \quad (8.44)$$

Equation (8.43) shows that the unknown temperature at grid point i and at time step $n+1$ is determined from the temperatures at the three grid points $i-1, i$, and $i+1$ which were computed and are known from the time step n . Such computational procedure can be described by the schematic diagram as shown in Fig. 8.18.

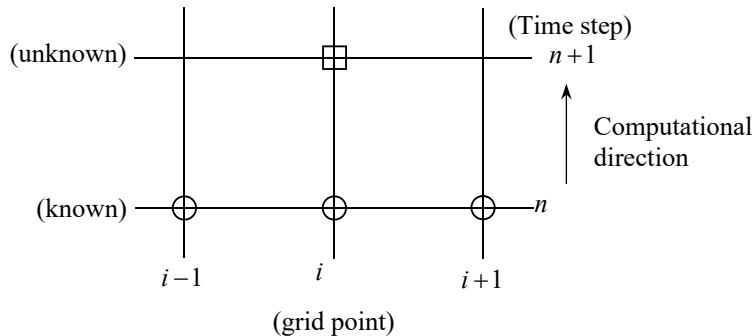


Figure 8.18 Schematic computational diagram of the explicit method.

Because the unknown temperature at time $n+1$ can be determined directly from the known temperatures at time n by using Eq. (8.43), the method is called the *explicit method*. It should be noted that even though the method is very convenient, it is limited by the value of $\alpha \leq 1/2$ which is governed by the time step Δt . In another word, the requirement of the time step,

$$\Delta t \leq \frac{\rho c (\Delta x)^2}{2k} \quad (8.45)$$

must be satisfied. The maximum allowable time step in Eq. (8.45) is called the *critical time step* which depends on the grid spacing Δx and the material properties. If the time step Δt used in the computation is greater than the critical time step, the computed solution will diverge or grow without bound. Because the time step Δt varies with the square of Δx , the time step will be very small for a fine mesh model with small Δx .

Example 8.4 Solve the parabolic Eq. (8.39) for the transient temperature distribution along the bar of length 1 unit. The bar is made from a material such that $k/\rho c = 1$ with the initial temperature of $\sin(\pi x)$ along its length. Determine the grid point temperatures by dividing the bar length into 10 equal intervals ($\Delta x = 0.1$).

The problem statement of this example can be summarized as follows:

$$\frac{k}{\rho c} \frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t} \quad 0 \leq x \leq 1 \quad (8.46)$$

$$T(0, t) = T(1, t) = 0 \quad (8.47)$$

$$T(x, 0) = \sin(\pi x) \quad (8.48)$$

If $k/\rho c = 1$, the exact solution is

$$T(x, t) = e^{-\pi^2 t} \sin(\pi x) \quad (8.49)$$

Figure 8.19 shows the bar dividing into 10 equal intervals with grid point numbers. Each interval has the length of $\Delta x = 0.1$. There are 11 grid points for which the temperatures at grid point numbers 1 and 11 are maintained at zero degree throughout the computation.

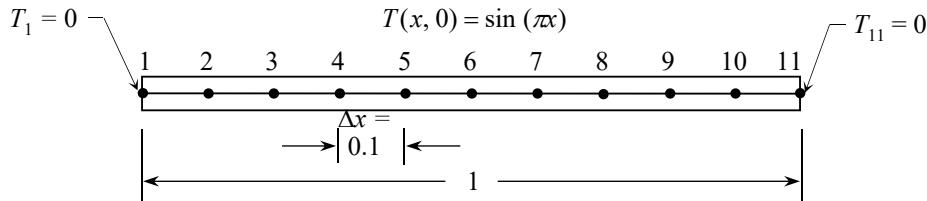


Figure 8.19 A bar with 10 equal intervals and grid point numbers.

The transient temperatures for grid point numbers 2 through 10 can be determined by using the derived Eq. (8.43). For example, the new temperature at grid point 2 is determined from the old temperatures at grid points 1, 2 and 3, as,

$$T_2^1 = T_2^0 + \alpha(T_3^0 - 2T_2^0 + T_1^0) \quad (8.50)$$

where the subscripts denote the grid point numbers and the superscripts represent the time step. Thus, the terms on the right-hand side of the equation are,

T_1^0 = temperature at grid point 1 at time $t = 0$ which is = 0

T_2^0 = temperature at grid point 2 at time $t = 0$ which is = $\sin(0.1\pi) = 0.3090$

T_3^0 = temperature at grid point 3 at time $t = 0$ which is = $\sin(0.2\pi) = 0.5878$

If the time step is specified as $\Delta t = 0.005$, then, from Eq. (8.44)

$$\alpha = \frac{k}{\rho c} \frac{\Delta t}{(\Delta x)^2} = (1) \frac{0.005}{(0.1)^2} = 0.5 \quad (8.51)$$

Thus, the temperature at grid point 2 at time $t = 0.005$ from Eq. (8.50) is,

$$T_2^1 = 0.3090 + (0.5)[0.5878 - 2(0.3090) + 0] = 0.2939$$

The computational procedure for determining transient temperatures at grid points can be used for developing a computer program. Figure 8.20 shows a computer program for determining the transient temperature response as explained in the example. The time step Δt and the total number of steps are specified at the beginning of the program. The temperatures at grid points are determined and are shown as output at each time step.

With the time step of $\Delta t = 0.005$, Table 8.3 shows the computed grid point temperatures at different times as compared to the exact solutions of Eq. (8.49). The table shows the computed temperatures for the grid points only on the left-half of the bar due to symmetry of the solution. Figure 8.20 shows the transient temperature response along the bar at different times. The bar temperature drops from the initial temperature of the specified sine function as time increases. As time increases, the bar temperature approaches zero because the temperature at both ends of the bar are maintained at zero.

```

% Program ParaExp1
% A finite difference program for solving
% transient temperature distribution in a
% rod using the explicit method.
% Assign time step and number of steps:
dt = 0.005; nsteps = 40;
% Set up initial and boundary conditions:
x = 0.; dx = 0.1;
for i = 1:11
    told(i) = sin(pi*x);
    tnew(i) = told(i);
    x = x + dx;
end
% Solve for the temperature response:
alpha = dt/(dx^2); time = dt;
for istep = 1:nsteps
    for i = 2:10
        tnew(i) = told(i) + alpha*(told(i+1) ...
            - 2.*told(i) + told(i-1));
    end
    % Print out the solutions at every 4 steps:
    ip = istep - floor(istep/4)*4;
    if ip == 0.
        fprintf('\n%6.2f',time);
        for i = 1:11
            fprintf(' %6.4f',tnew(i));
        end
    end
    for i = 2:10
        told(i) = tnew(i);
    end
    time = time + dt;
end

```

Figure 8.20 A computer program for determining transient temperature response of a bar by using the explicit method.

Table 8.3 Comparison between the computed temperatures at grid points using time step of $\Delta t = 0.005$ from the explicit method and the exact temperatures (values in brackets) at different times.

Time t	1	2	3	4	5	6
0.04	0.0000 (0.0000)	0.2068 (0.2082)	0.3934 (0.3961)	0.5415 (0.5451)	0.6366 (0.6408)	0.6693 (0.6738)
0.08	0.0000 (0.0000)	0.1384 (0.1403)	0.2633 (0.2669)	0.3625 (0.3673)	0.4261 (0.4318)	0.4480 (0.4540)
0.12	0.0000 (0.0000)	0.0927 (0.0945)	0.1763 (0.1798)	0.2426 (0.2475)	0.2852 (0.2910)	0.2999 (0.3059)
0.16	0.0000 (0.0000)	0.0620 (0.0637)	0.1180 (0.1212)	0.1624 (0.1668)	0.1909 (0.1961)	0.2007 (0.2062)
0.20	0.0000 (0.0000)	0.0415 (0.0429)	0.0790 (0.0816)	0.1087 (0.1124)	0.1278 (0.1321)	0.1344 (0.1389)

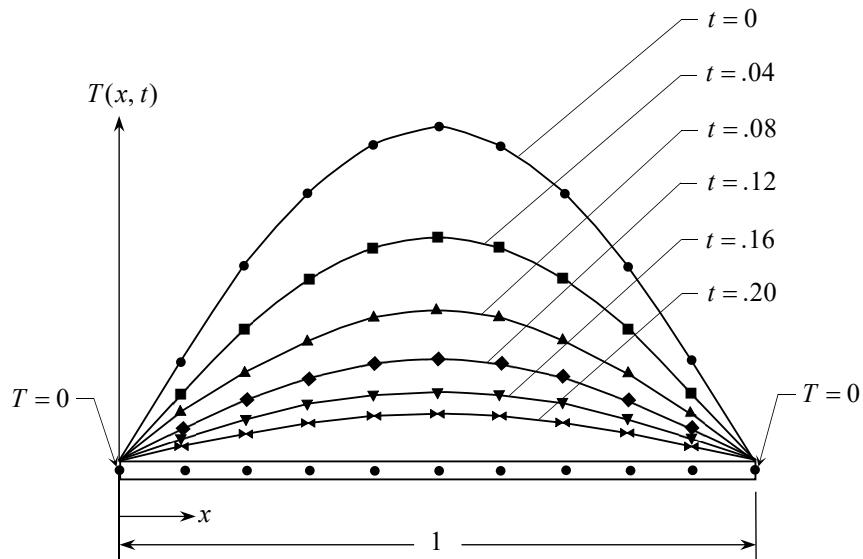


Figure 8.21 Computed transient temperature response at different times by using the explicit method.

It should be noted that the time step of $\Delta t = 0.005$ used to obtain the solutions is the critical time step for the mesh in the example. Higher solution accuracy can be obtained by reducing the time step Δt used in the computation. Table 8.4 shows the more accurate temperatures at the same grid points computed by using a smaller time step of $\Delta t = 0.00002$. Table 8.4 also shows the diverged solutions of the computed temperatures when the time step of $\Delta t = 0.01$ (which is larger than the critical time step) is used. Such diverged solution from using too large time step Δt is the main disadvantage of the explicit method. Other methods that can alleviate the restriction of too small time step Δt are presented in the following sections.

Table 8.4 Comparison among the computed temperatures at grid points using time step of $\Delta t = 0.00002$, $\Delta t = 0.01$ (values in square brackets) from the explicit method and the exact temperatures (values in round brackets) at different times.

Time t	1	2	3	4	5	6
0.04	0.0000 (0.0000)	0.2089 (0.2082)	0.3973 (0.3961)	0.5469 (0.5451)	0.6429 (0.6408)	0.6760 (0.6738)
	[0.0000]	[0.2047]	[0.3893]	[0.5358]	[0.6299]	[0.6623]
0.08	0.0000 (0.0000)	0.1412 (0.1403)	0.2686 (0.2669)	0.3697 (0.3673)	0.4346 (0.4318)	0.4570 (0.4540)
	[0.0000]	[0.1355]	[0.2578]	[0.3549]	[0.4171]	[0.4386]
0.12	0.0000 (0.0000)	0.0955 (0.0945)	0.1816 (0.1798)	0.2499 (0.2475)	0.2938 (0.2910)	0.3089 (0.3059)
	[0.0000]	[0.0900]	[0.1703]	[0.2356]	[0.2756]	[0.2913]
0.16	0.0000 (0.0000)	0.0645 (0.0637)	0.1227 (0.1212)	0.1689 (0.1668)	0.1986 (0.1961)	0.2088 (0.2062)
	[0.0000]	[0.0751]	[0.0829]	[0.1986]	[0.1295]	[0.2531]
0.20	0.0000 (0.0000)	0.0436 (0.0429)	0.0830 (0.0816)	0.1142 (0.1124)	0.1342 (0.1321)	0.1412 (0.1389)
	[0.0000]	[1.2042]	[-2.1852]	[3.3179]	[-3.8300]	[4.5065]

8.3.3 Implicit method

In order to avoid the diverged solution that may occur from using too large time step Δt in the explicit method, an implicit method is presented in this section. The implicit method applies the central difference approximation to the second-order spatial derivative term while the nodal temperatures are determined at time $n+1$ as,

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2} \quad (8.52)$$

By substituting Eqs. (8.52) and (8.40) into the parabolic differential Eq. (8.39), a corresponding algebraic equation is obtained,

$$\frac{k}{\rho c} \frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2} = \frac{T_i^{n+1} - T_i^n}{\Delta t} \quad (8.53)$$

which can be written in the form,

$$-\alpha T_{i-1}^{n+1} + (1+2\alpha) T_i^{n+1} - \alpha T_{i+1}^{n+1} = T_i^n \quad (8.54)$$

where the parameter α is defined in Eq. (8.44). The algebraic equation in Eq. (8.54) consists of the unknown temperatures at grid points at $i-1$, i and $i+1$ at the new time step $n+1$ on the left-hand side of the equation. The schematic computational diagram is shown in Fig. 8.22.

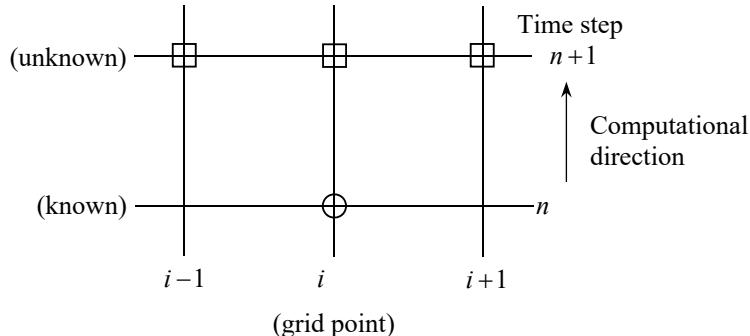


Figure 8.22 Schematic computational diagram of the implicit method.

Application of Eq. (8.54) to the grid points with unknown temperatures leads to a tridiagonal system of algebraic equations. The algebraic equations are coupled and the implicit method thus requires more computational time than that for the explicit method. However, the implicit method does not have restriction of the time step Δt used in the computation, i.e., the method will not produce a diverged solution. But a large time step Δt does produce inaccurate solution. Thus, analysts should understand the solution behaviors obtained from these methods. The example below shows the solution behavior obtained from the example of transient conduction heat transfer by using the implicit method.

Example 8.5 Solve the temperature response of Example 8.4 again but by using the implicit method. Use the time step of $\Delta t = 0.01$, which is larger than the critical time step of the problem which is $\Delta t = 0.005$. Compare the computed transient temperatures with the exact solution for $0 < t < 0.2$.

By using the time step of $\Delta t = 0.01$, Eq. (8.44) gives

$$\alpha = \frac{k}{\rho c} \frac{\Delta t}{(\Delta x)^2} = (1) \frac{0.01}{(0.1)^2} = 1 \quad (8.55)$$

The derived difference Eq. (8.54) with $\alpha = 1$ is

$$-T_{i-1}^{n+1} + 3T_i^{n+1} - T_{i+1}^{n+1} = T_i^n \quad (8.56)$$

Equation (8.54) is applied to all the grid points 2 to 10 for which their temperatures are unknowns. The application leads to a tridiagonal system of equations in the form,

$$\begin{bmatrix} 3 & -1 & & & \\ -1 & 3 & -1 & & \\ & -1 & 3 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 3 & -1 \\ & & & & -1 & 3 \end{bmatrix} \begin{Bmatrix} T_2 \\ T_3 \\ T_4 \\ \vdots \\ T_9 \\ T_{10} \end{Bmatrix}^{n+1} = \begin{Bmatrix} T_2 \\ T_3 \\ T_4 \\ \vdots \\ T_9 \\ T_{10} \end{Bmatrix}^n \quad (8.57)$$

It is noted that the boundary conditions of zero temperatures at grid points 1 and 11 have been imposed into the above trigonal system of Eq. (8.57). Equation (8.57) is then solved for the grid point temperatures at time step $n+1$.

Figure 8.23 shows the computer program for solving the temperature response at grid points by using the implicit method. The program is similar to that of the explicit method except the set of coupled algebraic equations are solved from a subroutine for a tridiagonal system of equations.

```
% Program ParaImp1
% A finite difference program for solving
% transient temperature distribution in a
% rod using the implicit method.
% Assign time step and number of steps:
dt = 0.01; nsteps = 20;
% Set up initial and boundary conditions:
x = 0.; dx = 0.1;
for i = 1:11
    temp(i) = sin(pi*x);
    x = x + dx;
end
alpha = dt/(dx^2); coef = 1 + 2*alpha;
time = dt; n = 9;
for istep = 1:nsteps
% Form up tridiagonal system of n eqs
% for the interior grids (here n=9):
b(1)=coef; c(1)=-alpha; d(1)=temp(2);
for i = 2:n-1
    a(i) = -alpha; b(i) = coef;
    c(i) = -alpha; d(i) = temp(i+1);
end
a(n)=-alpha; b(n)=coef; d(n)=temp(n+1);
% Solve such tridiagonal system of n eqs
% for temperatures at the interior grids.
% Return the solution in e( ):
for i = 2:n
    a(i) = a(i)/b(i-1);
    b(i) = b(i) - a(i)*c(i-1);
end
for i = 2:n
    d(i) = d(i) - a(i)*d(i-1);
end
e(n) = d(n)/b(n);
for i = n-1:-1:1
    e(i) = (d(i) - c(i)*e(i+1))/b(i);
end
for i = 2:10
    temp(i) = e(i-1);
end
% Print out the solution at every step:
ip = istep - floor(istep/2)*2;
if ip == 0.
    fprintf('\n%6.2f',time);
    for i = 1:11
        fprintf(' %6.4f',temp(i));
    end
end
time = time + dt;
end
```

Figure 8.23 A computer program for determining transient temperature response of a bar by using the implicit method.

The computed transient temperatures at grid points are compared with the exact solution as shown in Table 8.5. The table shows that the implicit method does not produce diverged solutions even though the time step Δt used in the computation is larger than the critical time step of the problem. However, the computed solutions at grid points contain large errors as compared to those obtained from the explicit method. This is mainly due to too large time step Δt is used in the analysis. It should be noted that both methods have the first-order of accuracy in time $O(\Delta t)$ and second-order of accuracy in space $O(\Delta x^2)$. In the next section, another implicit method that has the second-order of accuracy in both time and space is presented.

Table 8.5 Comparison between the computed temperatures at grid points using time step of $\Delta t = 0.01$ from the implicit method and the exact temperatures (values in brackets) at different times.

Time t	1	2	3	4	5	6
0.04	0.0000 (0.0000)	0.2127 (0.2082)	0.4046 (0.3961)	0.5568 (0.5451)	0.6546 (0.6408)	0.6883 (0.6738)
0.08	0.0000 (0.0000)	0.1464 (0.1403)	0.2785 (0.2669)	0.3833 (0.3673)	0.4506 (0.4318)	0.4737 (0.4540)
0.12	0.0000 (0.0000)	0.1008 (0.0945)	0.1917 (0.1798)	0.2638 (0.2475)	0.3101 (0.2910)	0.3261 (0.3059)
0.16	0.0000 (0.0000)	0.0694 (0.0637)	0.1319 (0.1212)	0.1816 (0.1668)	0.2134 (0.1961)	0.2244 (0.2062)
0.20	0.0000 (0.0000)	0.0477 (0.0429)	0.0908 (0.0816)	0.1250 (0.1124)	0.1469 (0.1321)	0.1545 (0.1389)

8.3.4 Crank-Nicolson method

The Crank-Nicolson method provides solution with second-order solution accuracy in both space and time. The method starts from approximating the first-order time derivative term in the form of the difference between the solutions at time steps n and $n+1$ as,

$$\frac{\partial T}{\partial t} = \frac{T_i^{n+1} - T_i^n}{\Delta t} \quad (8.58)$$

The method approximates the second-order spatial derivative term using,

$$\frac{\partial^2 T}{\partial x^2} = \frac{1}{2} \left(\frac{T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1}}{(\Delta x)^2} + \frac{T_{i+1}^n - 2T_i^n + T_{i-1}^n}{(\Delta x)^2} \right) \quad (8.59)$$

By substituting Eqs. (8.58) and (8.59) into the parabolic differential Eq. (8.37), the algebraic equation representing transient heat conduction in a bar is obtained as,

$$\frac{k}{2\rho c (\Delta x)^2} (T_{i+1}^{n+1} - 2T_i^{n+1} + T_{i-1}^{n+1} + T_{i+1}^n - 2T_i^n + T_{i-1}^n) = \frac{T_i^{n+1} - T_i^n}{\Delta t} \quad (8.60)$$

With the parameter α defined in Eq. (8.44), the above equations reduces to,

$$-\alpha T_{i-1}^{n+1} + 2(1+\alpha) T_i^{n+1} - \alpha T_{i+1}^{n+1} = \alpha T_{i-1}^n + 2(1-\alpha) T_i^n + \alpha T_{i+1}^n \quad (8.61)$$

The left-hand side of Eq. (8.61) consists of the unknown temperatures at grid points $i-1$, i and $i+1$ at the new time step $n+1$, while the right-hand side contains the known temperatures at time step n . The schematic computational diagrams corresponding to Eq. (8.61) is shown in Fig 8.24.

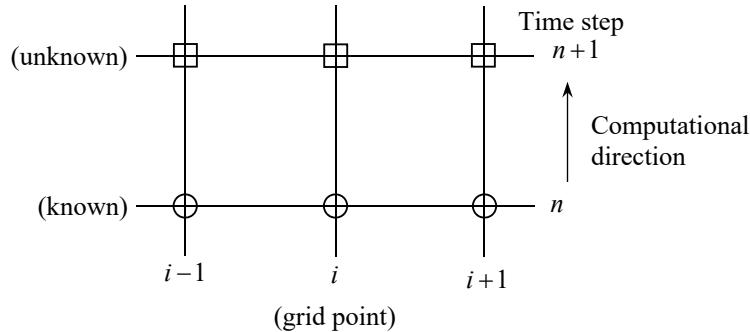


Figure 8.24 Schematic computational diagram of the Crank-Nicolson method.

Equation (8.59) is applied to all grid points for which the temperatures are unknown. The application leads to a tridiagonal set of equations that can be solved for the temperature solutions.

Example 8.6 Use the Crank-Nicolson method to determine the transient temperature response at grid points as described in Example 8.4. Employ the time step of $\Delta t = 0.02$ for $0 < t < 0.2$. Compare the computed grid point temperatures with the exact solution.

By using the time step $\Delta t = 0.02$, the parameter α from Eq. (8.42) is,

$$\alpha = \frac{k}{\rho c} \frac{\Delta t}{(\Delta x)^2} = (1) \frac{0.02}{(0.1)^2} = 2 \quad (8.62)$$

Then, Eq. (8.59) becomes,

$$-2T_{i-1}^{n+1} + 6T_i^{n+1} - 2T_{i+1}^{n+1} = 2T_{i-1}^n - 2T_i^n + 2T_{i+1}^n \quad (8.63)$$

Equation (8.63) is applied to grid points 2 to 10 for which their temperatures are unknown. The application leads to a tridiagonal set of equations similar to that shown in Eq. (8.57). The diagonal set of equations is then solved for the temperature solutions for the grid points at the new time step $n+1$.

The corresponding computer program of the Crank-Nicolson method for solving transient temperature response in the bar is shown in Fig. 8.25. The program is slightly more complicated than that for the implicit method. The computed temperature solutions at grid points, however, are more accurate. Table 8.6 shows the comparison of the computed temperature solutions obtained from the Crank-Nicolson method and the exact solutions for grid points and at 5 different times. Because the Crank-Nicolson method can provide higher solution accuracy as compared to the explicit and implicit methods, the method is widely used for analyzing transient problems in scientific and engineering applications.

```

% Program ParaCN1
% A finite difference program for solving
% transient temperature distribution in a
% rod using the Crank-Nicolson method.
% Assign time step and number of steps:
dt = 0.02; nsteps = 10;
% Set up initial and boundary conditions:
x = 0; dx = 0.1;
for i = 1:11
    temp(i) = sin(pi*x);
    x = x + dx;
end
alpha = dt/(dx^2); time = dt; n = 9;
coefp = 2*(1+alpha); coefm = 2*(1-alpha);
for istep = 1:nsteps
    % Form up tridiagonal system of n eqs
    % for the interior grids (here n=9):
    b(1) = coefp; c(1) = -alpha;
    d(1) = coefm*temp(2) + alpha*temp(3);
    for i = 2:n-1
        a(i)=-alpha; b(i)=coefp; c(i)=-alpha;
        d(i)= alpha*temp(i) + coefm*temp(i+1) ...
            + alpha*temp(i+2);
    end
    a(n) = -alpha; b(n) = coefp;
    d(n) = alpha*temp(n) + coefm*temp(n+1);
    % Solve such tridiagonal system of n eqs
    % for temperatures at the interior grids.
    % Return the solution in e( ):
    for i = 2:n
        a(i) = a(i)/b(i-1);
        b(i) = b(i) - a(i)*c(i-1);
    end
    for i = 2:n
        d(i) = d(i) - a(i)*d(i-1);
    end
    e(n) = d(n)/b(n);
    for i = n-1:-1:1
        e(i) = (d(i) - c(i)*e(i+1))/b(i);
    end
    for i = 2:10
        temp(i) = e(i-1);
    end
    % Print out the solution at every step:
    ip = istep - (istep/1)*1;
    if ip == 0.
        fprintf('\n%6.2f   ',time);
        for i = 1:11
            fprintf(' %6.4f   ',temp(i));
        end
    end
    time = time + dt;
end

```

Figure 8.25 A computer program for determining transient temperature response of a bar by using the Crank-Nicolson method.

Table 8.6 Comparison between the computed temperatures at grid points using time step of $\Delta t = 0.02$ from the Crank-Nicolson method and the exact temperatures (values in brackets) at different times.

time t	1	2	3	4	5	6
0.04	0.0000 (0.0000)	0.2086 (0.2082)	0.3968 (0.3961)	0.5462 (0.5451)	0.6421 (0.6408)	0.6752 (0.6738)
0.08	0.0000 (0.0000)	0.1409 (0.1403)	0.2679 (0.2669)	0.3688 (0.3673)	0.4335 (0.4318)	0.4558 (0.4540)
0.12	0.0000 (0.0000)	0.0951 (0.0945)	0.1809 (0.1798)	0.2490 (0.2475)	0.2927 (0.2910)	0.3078 (0.3059)
0.16	0.0000 (0.0000)	0.0642 (0.0637)	0.1221 (0.1212)	0.1681 (0.1668)	0.1976 (0.1961)	0.2078 (0.2062)
0.20	0.0000 (0.0000)	0.0434 (0.0429)	0.0825 (0.0816)	0.1135 (0.1124)	0.1334 (0.1321)	0.1403 (0.1389)

In order to summarize the solution accuracy obtained from the explicit, implicit and Crank-Nicolson methods in Examples 8.4, 8.5 and 8.6, respectively, their computed temperature solutions at the bar center obtained from using different time steps Δt are compared. The solutions are also compared with the exact solutions at time $t = 0.2$ as shown in Table 8.7. The table shows that the explicit method yields large solution errors even though the method provides an advantage for solving each equation explicitly. The method produces diverged solutions if the time step Δt used in the computation is larger than the critical time step, e.g., when $\alpha = 1.0$ as shown in the Table. The implicit method does not produce any diverged solution but the computed solutions are not accurate. As shown in the Table, the Crank-Nicolson method provides accurate solutions for different time steps Δt . As the time step Δt decreases, the method yields a solution that converges to the value of 0.1412. It is noted that the exact temperature solution at the middle of the bar is 0.1389. The difference between the Crank-Nicolson and exact solutions is from the number of grid points used for modeling the bar. If the number of grid points is increased by reducing the interval length Δx , then the computed solutions from the Crank-Nicolson

method will be close to the exact solution. Such numerical experiments for solution convergence of these methods to the exact solutions are left as exercises.

Table 8.7 Comparison of the computed temperatures at the bar center at time $t = 0.2$ from the three methods by using different time steps Δt . Note that the exact solution is 0.1389.

Time step Δt	α	Explicit	Implicit	Crank-Nicolson
0.0100	1.00	4.5065	0.1545	0.1410
0.0050	0.50	0.1344	0.1479	0.1411
0.0010	0.10	0.1398	0.1425	0.1412
0.0005	0.05	0.1405	0.1419	0.1412
0.0001	0.01	0.1410	0.1413	0.1412

Example 8.7 Use the explicit, implicit and Crank-Nicolson methods to solve the parabolic differential equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad (8.64)$$

for $0 \leq x \leq 1$ and $t > 0$ with the boundary conditions of

$$u(0,t) = 1 \quad \text{and} \quad u(1,t) = 0 \quad (8.65)$$

and the initial condition of

$$u(x,0) = 1 - x - \frac{1}{\pi} \sin(2\pi x) \quad (8.66)$$

Divide the domain into 20 equal intervals. Compare the computed solutions for $0 \leq t \leq 0.04$ by using $\Delta t = 0.002$ with the exact solution of

$$u(x,t) = 1 - x - \frac{1}{\pi} \sin(2\pi x) e^{-4\pi^2 t} \quad (8.67)$$

Figure 8.26 shows the computer program **ParaExp2** to compute the solution by using the explicit method. The program also displays the computed and exact solutions by the surface plots as shown in Fig. 8.27. User may want to change size of the interval Δx or the time step Δt to investigate the computed solution accuracy. The program can be modified to solve similar problems where their exact solutions are not available.

```
% Program ParaExp2
% A finite difference program for solving
% parabolic partial differential equation
% by using the explicit method.
% Assign the number of intervals, steps,
% domain length and total time:
ne = 20; nsteps = 20; xl = 1.; time = .04;
dx = xl/ne; np = ne + 1; dt = time/nsteps;
% Set up initial and boundary conditions:
x = 0:dx:xl; t = 0.;
uold = 1.-x-(1/pi).*sin(2*pi*x);
iplot = 1; uplot(:,iplot) = uold;
% Solve for solutions at each time step:
alpha = dt/(dx^2); t = t + dt; iplot = 2;
for istep = 1:nsteps
    for i = 2:ne
        unew(i) = uold(i) + alpha*(uold(i+1) ...
            - 2.*uold(i) + uold(i-1));
    end
    % Surface plot of the exact solutions:
    xe = 0:dx:xl; te = 0:dt:time;
    [XE,TE] = meshgrid(xe,te);
    ueplot = 1.-XE-(1/pi).*sin(2*pi*XE).* ...
        exp(-4*pi*pi*TE);
    mesh(XE,TE,ueplot), view(30,30),
    colormap([0,0,0])
    % Surface plot of the computed solutions:
    xc = 0:dx:xl; tc = 0:dt:time;
    [XC,TC] = meshgrid(xc,tc); figure
    mesh(XC,TC,uplot'); view(30,30);
    colormap([0,0,0])
end
```

Figure 8.26 A computer program for determining transient response of $u(x,t)$ by using the explicit method for Example 8.7.

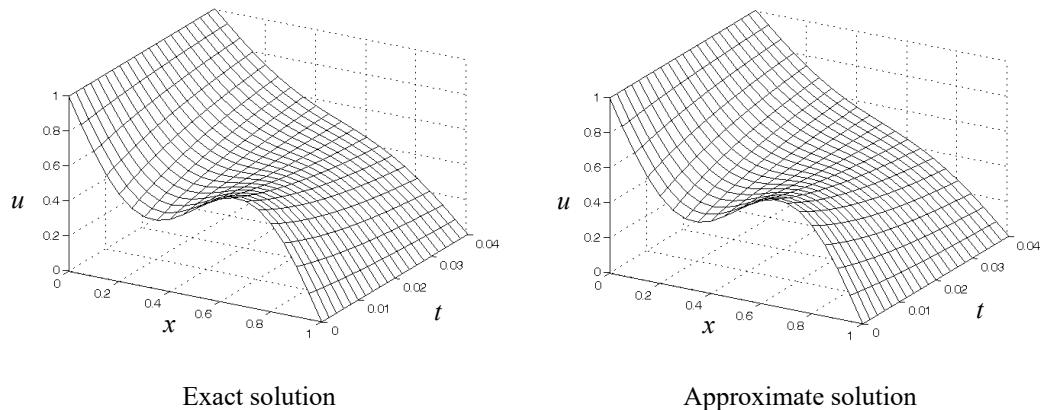


Figure 8.27 Comparison of the exact and explicit solutions of $u(x,t)$ for Example 8.7.

Figures 8.28 and 8.29 show the computer programs **ParaIMP2** and **ParaCN2** for determining the approximate solutions by using the implicit and Crank-Nicolson methods, respectively. The computed solutions are displayed in form of the surface plots. User may change the grid size Δx or time step Δt to study the accuracy of the computed solutions. It is noted that, with the same grid size and time step, the Crank-Nicolson method produces higher solution accuracy than the implicit method.

```
% Program ParaImp2
% A finite difference program for solving
% parabolic partial differential equation
% by using the implicit method.
% Assign the number of intervals, steps,
% domain length and total time:
ne = 20; nsteps = 20; xl = 1.; time = .04;
dx = xl/ne; np = ne + 1; dt = time/nsteps;
% Set up initial and boundary conditions:
x = 0:dx:xl; t = 0.;
u = 1.-x-(1/pi).*sin(2*pi*x);
u(1) = 1; u(np) = 0;
iplot = 1; uplot(:,iplot) = u;
% Solve for solutions at each time step:
t = t + dt; iplot = 2;
alpha = dt/(dx^2); coef = 1 + 2*alpha;
a = zeros(np,np); a(1,1) = 1; a(np,np) = 1;
d = zeros(np,1); d(1) = u(1); d(np) = u(np);
% Form up the system LHS of np equations:
for i = 2:np-1
    a(i,i) = coef;
    a(i,i-1) = -alpha; a(i,i+1) = -alpha;
end
% March out for solution at each step for
% the total of n steps:
for istep = 1:nsteps
    for i = 2:np-1
        d(i) = u(i);
    end
    % Solve the system of np equations for
    % solution at each step, return solution
    % in e( ), and save them for plotting:
    e = a\d; u = e; uplot(:,iplot) = u;
    t = t + dt; iplot = iplot + 1;
end
% Surface plot of the exact solutions:
xe = 0:dx:xl; te = 0:dt:time;
[XE,TE] = meshgrid(xe,te);
ueplot = 1.-XE-(1/pi).*sin(2*pi*XE).* ...
    exp(-4*pi*pi*TE);
mesh(XE,TE,ueplot), view(30,30),
colormap([0,0,0])
% Surface plot of the computed solutions:
xc = 0:dx:xl; tc = 0:dt:time;
[XC,TC] = meshgrid(xc,tc); figure
mesh(XC,TC,uplot'); view(30,30);
colormap([0,0,0])
```

Figure 8.28 A computer program for determining transient response of $u(x,t)$ by using the implicit method for Example 8.7.

```

% Program ParaCN2
% A finite difference program for solving
% parabolic partial differential equation
% by using the Crank-Nicolson method.
% Assign the number of intervals, steps,
% domain length and total time:
ne = 20; nsteps = 20; xl = 1.; time = .02;
dx = xl/ne; np = ne + 1; dt = time/nsteps;
% Set up initial and boundary conditions:
x = 0:dx:xl; t = 0.;
u = 1.-x-(1/pi).*sin(2*pi*x);
u(1) = 1; u(np) = 0;
iplot = 1; uplot(:,iplot) = u;
% Solve for solutions at each time step:
t = t + dt; iplot = 2; alpha = dt/(dx^2);
coefp = 2*(1+alpha); coefm = 2*(1-alpha);
a = zeros(np,np); a(1,1) = 1; a(np,np) = 1;
d = zeros(np,1); d(1) = u(1); d(np) = u(np);
% Form up the system LHS of np equations:
for i = 2:np-1
    a(i,i) = coefp;
    a(i,i-1) = -alpha; a(i,i+1) = -alpha;
end
% March out for solution at each step for
% the total of n steps:
for istep = 1:nsteps
    for i = 2:np-1
        d(i) = alpha*u(i-1) + coefm*u(i) + ...
            alpha*u(i+1);
    end
    % Solve the system of np equations for
    % solution at each step, return solution
    % in e(), and save them for plotting:
    e = a\;d; u = e; uplot(:,iplot) = u;
    t = t + dt; iplot = iplot + 1;
end
% Surface plot of the exact solutions:
xe = 0:dx:xl; te = 0:dt:time;
[XE,TE] = meshgrid(xe,te);
ueplot = 1.-XE-(1/pi).*sin(2*pi*XE).* ...
    exp(-4*pi*pi*TE);
mesh(XE,TE,ueplot), view(30,30),
colormap([0,0,0])
% Surface plot of the computed solutions:
xc = 0:dx:xl; tc = 0:dt:time;
[XC,TC] = meshgrid(xc,tc); figure
mesh(XC,TC,uplot'), view(30,30),
colormap([0,0,0])

```

Figure 8.29 A computer program for determining transient response of $u(x,t)$ by using the Crank-Nicolson method for Example 8.7.

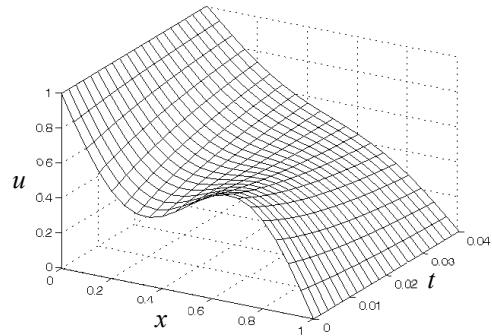
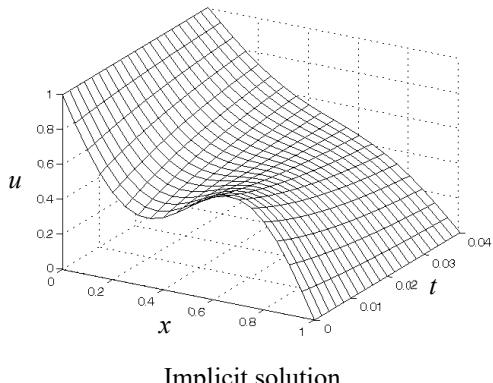


Figure 8.30 The implicit and Crank-Nicolson solutions of $u(x,t)$ for Example 8.7.

8.4 Hyperbolic Equation

8.4.1 Differential equation

Hyperbolic equation is the partial differential equation that normally describes propagation behaviors from one point to another. Examples of such propagation behaviors are the oscillation of a string fixed at both ends, shock wave propagation that occurs in a tube from different air densities, traveling of a wave in a solid from an impact loading, etc. Many of these behaviors consist of sudden changes in the solution. Sudden changes in the solution require a fine mesh with small interval Δx to produce accurate solution. Such accurate solution are thus difficult to obtain, so many realistic

problems that are governed by the hyperbolic equation are still under research development. In this section, a hyperbolic equation that arises from a simple phenomenon is considered. Figure 8.31 shows the oscillation behavior of a string with length L which is fixed at both ends.

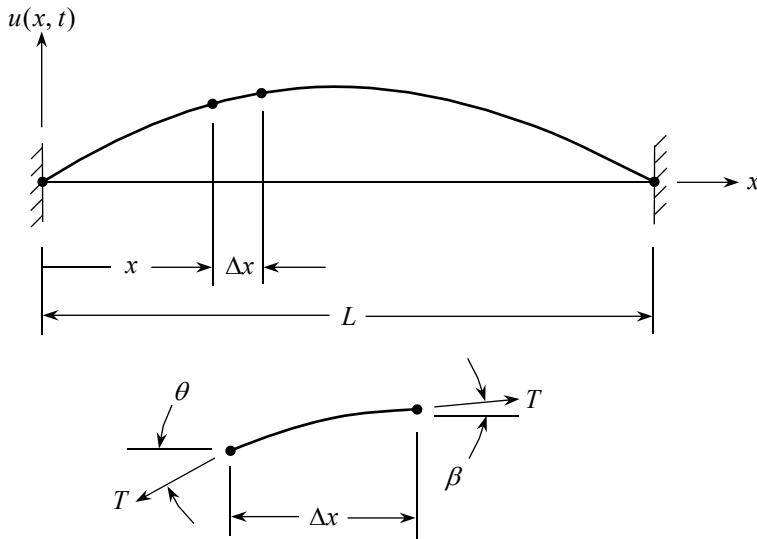


Figure 8.31 Oscillation behavior of a string and its portion under equilibrium.

The governing differential equation for the string deflection $u(x,t)$ that varies with the string x -coordinate and time t can be derived by considering the lower insert of Fig. 8.31. For a small oscillation, the deflecting angles θ and β are small while the string tension T can be assumed constant. By neglecting the weight of the string, the Newton's second law is applied to the small segment of the string to yield

$$T \sin \beta - T \sin \theta = \frac{w}{g} (\Delta x) \frac{\partial^2 u}{\partial t^2} \quad (8.68)$$

where w is the string weight per unit length and g is the gravitational acceleration constant. With the small angles θ and β , then $\sin \theta = \tan \theta$ and $\sin \beta = \tan \beta$, thus Eq. (8.68) becomes

$$T \tan \beta - T \tan \theta = \frac{w}{g} (\Delta x) \frac{\partial^2 u}{\partial t^2} \quad (8.69)$$

But $\tan \beta$ and $\tan \theta$ represent the slopes at the ends of the segment, Eq. (8.69) can be written as

$$\begin{aligned} T \left(\frac{\partial u}{\partial x} \Big|_{x+\Delta x} - \frac{\partial u}{\partial x} \Big|_x \right) &= \frac{w}{g} (\Delta x) \frac{\partial^2 u}{\partial t^2} \\ \text{Or, } \frac{\frac{\partial u}{\partial x} \Big|_{x+\Delta x} - \frac{\partial u}{\partial x} \Big|_x}{\Delta x} &= \frac{w}{Tg} \frac{\partial^2 u}{\partial t^2} \end{aligned} \quad (8.70)$$

Then, by letting $\Delta x \rightarrow 0$, Eq. (8.70) becomes,

$$\frac{\partial^2 u}{\partial x^2} = \frac{w}{Tg} \frac{\partial^2 u}{\partial t^2} \quad (8.71)$$

Or,

$$\frac{\partial^2 u}{\partial t^2} = k^2 \frac{\partial^2 u}{\partial x^2} \quad (8.72)$$

where

$$k^2 = \frac{Tg}{w} \quad (8.73)$$

It is noted that Eq. (8.72) is in the form of Eq. (8.9), which is the standard form of the hyperbolic equation in one dimension where k^2 is always positive.

8.4.2 Computational procedures

In this section, the finite difference method is applied to solve the hyperbolic equation. The string is first divided into intervals, with the length of Δx each. These intervals are connected by grid points at $i-1, i, i+1$ as shown in Fig. 8.32.

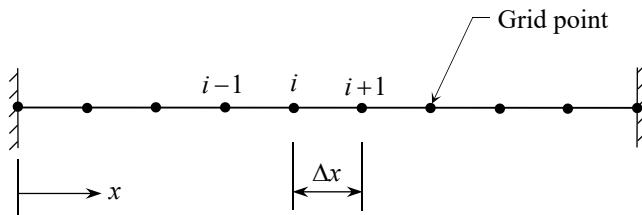


Figure 8.32 A string is divided into intervals connected at grid points.

At these grid points, their deflections u which depend on time t are to be determined. The central differencing is used to approximate the second-order time derivative term,

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{(\Delta t)^2} \quad (8.74)$$

where the subscript i represents the grid point number and the superscript n denotes the time step. The central differencing is also used to approximate the second-order spatial derivative term,

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2} \quad (8.75)$$

By substituting Eqs. (8.74) and (8.75) into Eq. (8.72), the algebraic equation representing the hyperbolic equation is obtained,

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{(\Delta t)^2} = k^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{(\Delta x)^2}$$

After rearranging terms, the deflection for grid point i at time step $n+1$ can be determined from

$$u_i^{n+1} = 2u_i^n - u_i^{n-1} + C(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \quad (8.76)$$

where

$$C = \frac{k^2(\Delta t)^2}{(\Delta x)^2} \quad (8.77)$$

The parameter C in Eq. (8.77) is called the Courant number. It has been found that accurate solution is obtained when the Courant number C is close to unity. If $C > 1$, the computed solution may diverge from the actual solution. If $C < 1$, the computed solution is less accurate. Study of different values of C that affect the solution accuracy is left as an exercise. With the Courant number as unity, appropriate time step Δt can be determined from the known grid point spacing Δx in the finite difference model.

Figure 8.33 shows the schematic computational diagram for the derived finite difference Eq. (8.76) representing the string oscillation. The unknown deflection for grid point i at time step $n+1$ is determined from the known deflections for grid points $i-1, i, i+1$ at time step n and the known deflection for grid point i at time step $n-1$.

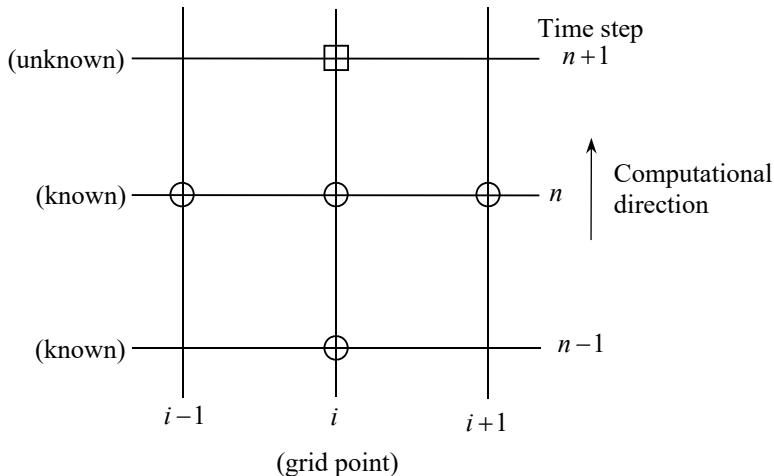


Figure 8.33 Schematic computational diagram for solving the hyperbolic equation.

At the beginning of the computational step 1 ($n = 0$), Eq. (8.76) needs the known solutions at steps 0 and -1 . The deflection at step -1 can be determined from the initial condition of the problem. For example, if the initial velocity is given as zero, i.e.,

$$\frac{\partial u}{\partial t}(x, t=0) = 0 \quad (8.78)$$

Then, the central difference approximation for Eq. (8.78) is,

$$\frac{u_i^1 - u_i^{-1}}{2(\Delta t)} = 0 \quad (8.79)$$

Or,

$$u_i^{-1} = u_i^1 \quad (8.80)$$

By substituting Eq. (8.74) into Eq. (8.70) at $n = 0$,

$$u_i^1 = 2u_i^0 - u_i^1 + C(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0)$$

Or,

$$u_i^1 = u_i^0 + \frac{C}{2}(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0) \quad (8.81)$$

In summary, the deflections at grid points along the string are first determined at $n = 0$ by using Eq. (8.81). Equation (8.76) is then used for determining the grid point deflections at the later time steps. Since the Courant number is assigned as unity, the deflections at grid points can be determined by using Eqs. (8.81) and (8.76) as follows,

$$u_i^1 = (u_{i+1}^0 + u_{i-1}^0)/2 \quad \text{when } n=0 \quad (8.82a)$$

$$u_i^{n+1} = -u_i^{n-1} + u_{i+1}^n + u_{i-1}^n \quad \text{when } n > 0 \quad (8.82b)$$

8.4.3 Examples

Two examples are presented in this section and corresponding computer programs are developed so that the computed solutions can be compared with the exact solutions.

Example 8.8 A string of length 1.5 unit is fixed at both ends. The string is released from the initial configuration as shown in Fig. 8.34. If the values $k = 100$, the grid spacing $\Delta x = 0.25$ and the Courant number $C = 1$, then the appropriate time step computed from Eq. (8.77) is $\Delta t = 0.0025$. Determine the string oscillation behavior for $0 \leq t \leq 0.03$. Compare the computed solutions at grid points with the exact solution of

$$u(x, t) = \frac{2aL^2}{b(L-b)\pi^2} \sum_{n=1}^{\infty} \frac{1}{n^2} \sin\left(\frac{n\pi}{L}\right) \cos\left(\frac{100n\pi t}{L}\right) \sin\left(\frac{n\pi x}{L}\right) \quad (8.83)$$

where a and b are the deflection and distance of the initial string configuration as shown in Fig. 8.34.

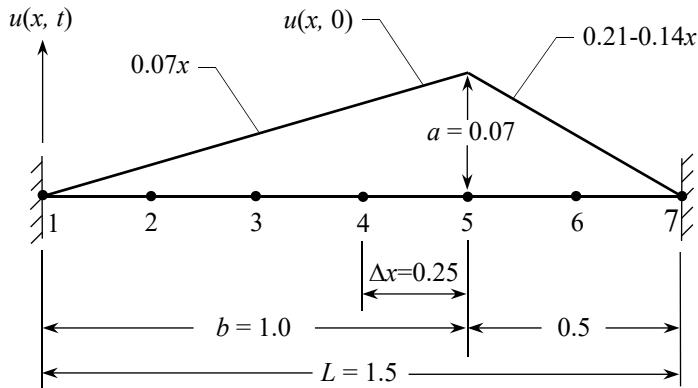


Figure 8.34 A string is divided into 6 intervals and 7 grid points with its initial configuration.

Equations (8.82a-b) are used to determine the string deflection for grid point i at any time step n . For example, the deflection for grid point 5 at the first time step ($t = 0.0025$) is determined from Eq. (8.82a) as,

$$\begin{aligned} u_5^1 &= (u_6^0 + u_4^0)/2 \\ &= (0.0350 + 0.0525)/2 = 0.04375 \end{aligned}$$

Similarly, the deflections for grid points 4 and 6 at the first time step are

$$u_4^1 = 0.0525 \quad \text{and} \quad u_6^1 = 0.0350$$

Then, the deflection for grid point 5 at the second time step ($t = 0.0050$) are determined by using Eq. (8.82b) as,

$$\begin{aligned} u_5^2 &= -u_5^0 + u_6^1 + u_4^1 \\ &= -0.07 + 0.0350 + 0.0525 = 0.0175 \end{aligned}$$

The computational procedure explained above is used to develop a computer program as shown in Fig. 8.35. The program starts from assigning the time step Δt and the total number of steps for determining the string oscillation behavior. After initial and boundary conditions are specified, the program employs Eq. (8.82a) to determine the grid point deflections at the first time step. Then, Eq. (8.82b) is used for determining the grid point deflections for the rest of the time steps.

```
% Program Hyper1
% A finite difference program for solving
% vibration in a string.
% Assign time step and number of time steps:
dtime = 0.0025; nsteps = 12;
% Set up initial and boundary conditions:
xx = 0.0; dx = 0.25; time = 0.0;
for i = 1:5
    un(i) = 0.07*xx; x(i) = xx; xx = xx + dx;
end
xx = 5*dx;
for i = 6:7
    un(i) = 0.21 - 0.14*xx; x(i) = xx;
    xx = xx + dx;
end
plot(x,un,'-ok'); hold on;
% Compute displacement at the first time
% step:
time = time + dtime; istep = 1;
unp1(1,istep) = un(1);
for i = 2:6
    unp1(i,istep) = 0.5*(un(i+1) + un(i-1));
end
unp1(7,istep) = un(7);
% Compute displacements at the second time
% step onward:
time = time + dtime;
for istep = 2:nsteps
    for i = 1:7
        unml(i) = un(i);
        un(i) = unp1(i,istep-1);
    end
    for i = 2:6
        unp1(i,istep) = -unml(i) + un(i+1) + un(i-1);
    end
    fprintf('\n%6.4f ', time);
    for i = 1:7
        fprintf(' %8.5f ',unp1(i,istep));
    end
    time = time + dtime;
end
% Plot the computed solutions at
% different times:
for istep = 1:nsteps
    plot(x,unp1(:,istep),'-ok');
    hold on
end
```

Figure 8.35 A computer program for determining deflection of a string fixed at both ends.

Table 8.8 shows the comparison between the computed deflections at grid points and the exact solution. In this example, the computed solutions are exact and the string oscillation behaviors at different times are shown in Fig. 8.36. The figure shows the string deflection shapes for the first half cycle $0 \leq t \leq 0.015$. The string deflection shapes are reversed for the second half cycle $0.015 \leq t \leq 0.030$. The deflection shape becomes the initial shape again at time $t = 0.030$ after one cycle of the oscillation is complete.

Table 8.8 Comparison between the computed deflections at grid points using time step of $\Delta t = 0.0025$ and the exact solutions (values in brackets) at different times. Note that the deflections at grid points 1 and 7 are zero.

Time t	Deflections at grid points				
	2	3	4	5	6
0.000	0.01750 (0.01750)	0.03500 (0.03500)	0.05250 (0.05250)	0.07000 (0.07000)	0.03500 (0.03500)
0.005	0.01750 (0.01750)	0.03500 (0.03500)	0.02625 (0.02625)	0.01750 (0.01750)	0.00875 (0.00875)
0.010	-0.00875 (-0.00875)	-0.01750 (-0.01750)	-0.02625 (-0.02625)	-0.03500 (-0.03500)	-0.01750 (-0.01750)
0.015	-0.03500 (-0.03500)	-0.07000 (-0.07000)	-0.05250 (-0.05250)	-0.03500 (-0.03500)	-0.01750 (-0.01750)
0.020	-0.00875 (-0.00875)	-0.01750 (-0.01750)	-0.02625 (-0.02625)	-0.03500 (-0.03500)	-0.01750 (-0.01750)
0.025	0.01750 (0.01750)	0.03500 (0.03500)	0.02625 (0.02625)	0.01750 (0.01750)	0.00875 (0.00875)
0.030	0.01750 (0.01750)	0.03500 (0.03500)	0.05250 (0.05250)	0.07000 (0.07000)	0.03500 (0.03500)

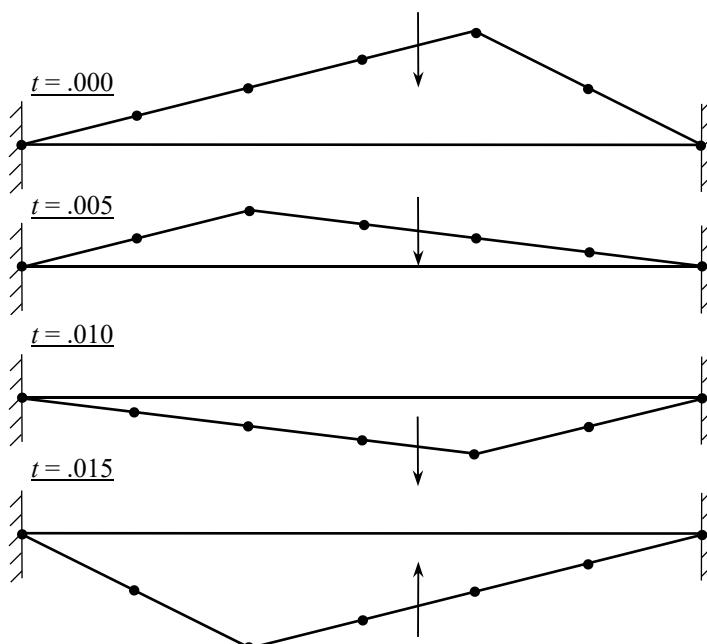


Figure 8.36 String deflection shapes at different times for $0 \leq t \leq 0.015$. The shapes are reversed for $0.015 \leq t \leq 0.030$ when the string deflects back to initial configuration at time 0.030.

Example 8.9 Develop a computer program to solve the hyperbolic differential equation in the form

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \quad (8.84)$$

for $0 \leq x \leq 1$ and $t > 0$ with the boundary conditions of

$$u(0,t) = 0 \quad \text{and} \quad u(1,t) = 0 \quad (8.85)$$

and the initial conditions of

$$u(x,0) = \sin(\pi x) \quad \text{and} \quad \frac{\partial u}{\partial t}(x,0) = 0 \quad (8.86)$$

Divide the domain into 20 equal intervals. Compare the computed solutions for $0 \leq t \leq 4$ by using a Courant number of one with the exact solution of

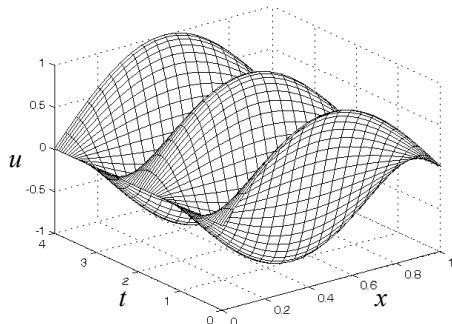
$$u(x,t) = \sin(\pi x) \cos(\pi t) \quad (8.87)$$

Figure 8.37 shows the computer program **Hyper2** to compute the solution $u(x,t)$ and compare with the exact solution. The program displays the computed and exact solutions in form of the surface plots as shown in Fig. 8.38. User may want to change size of the interval Δx or the time step Δt to study the computed solution accuracy.

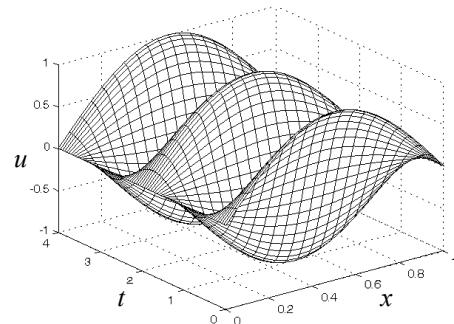
```
% Program Hyper2
% A finite difference program for solving
% hyperbolic partial differential equation.
% Assign the number of intervals,
% domain length and total time:
ne = 20; xl = 1.; time = 4;
dx = xl/ne; np = ne + 1;
% Determine the time step according to the
% Courant number of 1.
dt = dx; nsteps = time/dt;
% Set up initial and boundary conditions:
x = 0:dx:xl; t = 0.; un = sin(pi*x);
iplot = 1; uplot(:,iplot) = un;
% Compute solutions at the first time step:
istep = 1; t = t + dt; iplot = 2;
unpl(1) = un(1); uplot(1,iplot) = unpl(1);
for i = 2:np-1
    unpl(i) = (un(i+1) + un(i-1))/2.;
    uplot(i,iplot) = unpl(i);
end
unpl(np) = un(np);
uplot(np,iplot) = unpl(np);

% Compute solutions at second step onward:
t = t + dt; iplot = 3;
for istep = 2:nsteps
    for i = 1:np
        unml(i) = un(i); un(i) = unpl(i);
    end
    for i = 2:np-1
        unpl(i) = -unml(i) + un(i+1) + un(i-1);
        uplot(i,iplot) = unpl(i);
    end
    t = t + dt; iplot = iplot + 1;
end
% Surface plot of the exact solutions:
xe = 0:dx:xl; te = 0:dt:time;
[XE,TE] = meshgrid(xe,te);
ue = sin(pi*XE).*cos(pi*TE);
ueplot = fliplr(ue); figure
mesh(XE,TE,ueplot); colormap([0,0,0])
% Surface plot of the computed solutions:
xc = 0:dx:xl; tc = 0:dt:time;
[XC,TC] = meshgrid(xc,tc); figure
mesh(XC,TC,uplot'); colormap([0,0,0])
```

Figure 8.37 A computer program for determining the response $u(x,t)$ of the hyperbolic differential equation for Example 8.9.



Exact solution



Approximate solution

Figure 8.38 Comparison between the exact and approximate solutions of the hyperbolic differential equation in Example 8.9.

8.5 Closure

The finite difference method for solving the partial differential equations is presented in this chapter. The partial differential equations are classified into three types of elliptic, parabolic and hyperbolic equations. Appropriate boundary and initial conditions for different types of the partial differential equations were described. Simple examples associated with these types of equations are explained with their physical meanings and solution behaviors.

The elliptic equation is the simplest one among the three types of partial differential equations. The finite difference method for solving the elliptic equation is easy to understand. Steady-state heat conduction in a plate is an example that can be solved conveniently by using the finite difference method. The method divides the plate into a number of intervals where the unknowns are at the grid points connecting these intervals. The elliptic partial differential equation is transformed into an algebraic equation which applies at the grid points. Such application leads to a set of equations that can be solved by the Gauss-Seidel iteration method for the solutions at these grid points.

In the process for solving the parabolic equation, the example of transient conduction heat transfer in a bar was used. Three types of the finite difference methods, which are the explicit, the implicit and the Crank-Nicolson methods, were explained. Examples were presented to highlight the solution accuracy obtained from these methods. Among these methods, the Crank-Nicolson method provides higher solution accuracy than the other two methods. The Crank-Nicolson method is thus widely used in commercial software for solving the parabolic equation.

The hyperbolic equation is the most difficult differential equation for obtaining solutions as compared to the elliptic and parabolic equations. This is mainly because its solutions normally change suddenly within a narrow region. A large number of short intervals is thus needed to produce an accurate solution. Such large number of intervals leads to many unknowns at grid points. Solving the hyperbolic equation thus requires a large computational effort in both the computer time and memory.

All computational techniques presented in this chapter show the simplicity of the finite difference method for solving the partial differential equations. The corresponding computer programs can also be developed straightforwardly. Because the method divides a computational domain into intervals with rectangular shapes, the method is not suitable for problems that have complex geometry. The problems with complex geometry can be handled conveniently by using the finite element method presented in the following chapter.

Exercises

1. Identify each of the following partial differential equations whether it is the elliptic, parabolic or hyperbolic equation. Then, suggest appropriate computational procedure for solving each of them.
 - (a) A differential equation for inviscid compressible flow in two-dimensional x - y coordinates is governed by

$$(1 - M^2) \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

where M is the Mach number which is less than unity for the subsonic flow and ϕ is the velocity potential.

- (b) A one-dimensional unsteady Navier-Stokes equation is given by

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2}$$

where u is the velocity that varies with x -coordinate and time t , and ν is the fluid viscosity.

- (c) A differential equation which is in the form

$$(x^2 - 1) \frac{\partial^2 u}{\partial x^2} + 2y \frac{\partial^2 u}{\partial x \partial y} - \frac{\partial^2 u}{\partial y^2} = 0$$

where u is the dependent variable that varies with x and y .

- (d) A differential equation that governs a steady-state viscous flow is in the form

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = -\frac{c}{\nu}$$

where ϕ is the flow velocity that varies with x - and y -coordinates, c is the pressure difference in the flow direction and ν is the fluid viscosity.

- (e) A differential equation representing transient heat conduction in a plate with surface convection is

$$-a \frac{\partial^2 T}{\partial x^2} + b \frac{\partial T}{\partial x} + cT + \frac{\partial T}{\partial t} = 0$$

where T is the temperature and a, b, c are positive integers.

2. Use the finite difference method to solve the Laplace's equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad 0 \leq x \leq 1, 0 \leq y \leq 1$$

with the boundary conditions of

$$\begin{aligned} u(x, 0) &= 0, & u(x, 1) &= x, & 0 \leq x \leq 1 \\ u(0, y) &= 0, & u(1, y) &= y, & 0 \leq y \leq 1 \end{aligned}$$

Divide the unit square region into 3×3 intervals in both x - and y -directions. Compare the computed solutions at grid points with the exact solution of $u(x, y) = xy$.

3. Solve Problem 2 again but by dividing the unit square region into 100×100 intervals and developing a computer program. Compare the computed solutions at grid points with the exact solution.
4. Modify the computer program in Fig. 8.10 to solve the plate temperature distribution by dividing the plate into 200 and 100 intervals in x - and y -directions, respectively. Compare the computed solutions at grid points with the exact solution in Eq. (8.25).

5. Because the temperature distribution in Problem 4 has symmetry over the plate, thus only the left half of the plate can be used for modeling. Divide the left half of the plate into 100×100 intervals in both x - and y -directions. Compare the computed solutions at grid points with those obtained from the full model.
6. Consider Fig. 8.13 for constructing an algebraic equation of grid points along the insulated boundary where

$$\frac{\partial T}{\partial x} = 0$$

Then, develop a new algebraic equation when the boundary has surface convection to a surrounding medium temperature T_∞ where

$$\frac{\partial T}{\partial x} = \frac{T - T_\infty}{\Delta x}$$

Explain how to implement the derived algebraic equation to the computer program in Fig. 8.10.

7. Develop the algebraic equation for solving the Poisson's equation

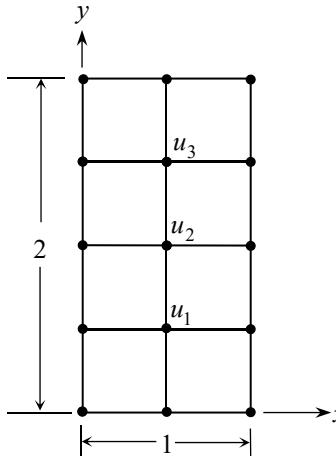


Figure P8.7.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 4 \quad 0 \leq x \leq 1, 0 \leq y \leq 2$$

with the boundary conditions of

$$u(x, 0) = x^2, u(x, 2) = (x-2)^2, \quad 0 \leq x \leq 1$$

$$u(0, y) = y^2, u(1, y) = (y-1)^2, \quad 0 \leq y \leq 2$$

Divide the domain into 2 and 4 intervals in x - and y -directions, respectively, as shown in Fig. P8.7. Compare the computed solutions at grid points with the exact solution of $u(x, y) = (x-y)^2$.

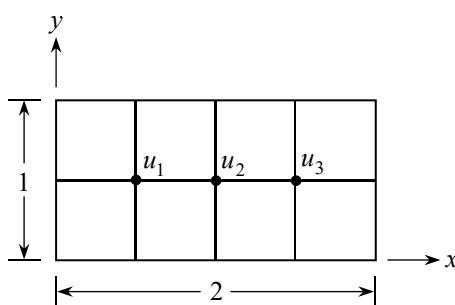
8. Solve Problem 7 again but by developing a computer program. Divide the domain into 50 and 100 intervals in x - and y -directions, respectively. Plot to compare the computed temperature distribution with the exact solution.
9. Steady-state heat conduction in a plate with internal heat generation is governed by the partial differential equation

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = -\frac{Q}{k}$$

where T is the temperature that varies with the plate x - y coordinates, Q is the internal heat generation rate per unit volume and k is the thermal conductivity coefficient of the plate material. Use the central differencing approximation to derive the algebraic equation in a stencil form similar to Eq. (8.19). Then, explain how to apply the stencil form to solve for the plate temperature distribution.

10. Use the central difference method to derive the algebraic equation corresponding to the Poisson's equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = (x^2 + y^2)e^{xy} \quad 0 \leq x \leq 2, 0 \leq y \leq 1$$



with the boundary conditions of

$$u(x, 0) = 1, u(x, 1) = e^x, \quad 0 \leq x \leq 2$$

$$u(0, y) = 1, u(2, y) = e^{2y}, \quad 0 \leq y \leq 1$$

Divide the domain into 4×2 intervals in x - and y -directions, respectively, as shown in Fig. P8.10. Compare the computed solutions with the exact solution of $u(x, y) = e^{xy}$.

Figure P8.10.

11. Solve Problem 10 again but by developing a computer program. Then use the program to determine the unknowns at grid points by dividing the domain into

- (a) 10 and 5 intervals in x - and y -directions, respectively,
- (b) 20 and 10 intervals in x - and y -directions, respectively,
- (c) 40 and 20 intervals in x - and y -directions, respectively,

Plot to compare the computed solutions obtained from each model with the exact solution. Then, give comments on the convergence of the computed solutions to the exact solution as the model is refined by using more intervals.

12. Use the central difference method to derive the algebraic equation corresponding to the Poisson's equation in Problem 10 but with different interval lengths of Δx and Δy . Then, implement the derived equation on a computer program to solve the same problem by using

- (a) $\Delta x = 0.2$ and $\Delta y = 0.1$
- (b) $\Delta x = 0.1$ and $\Delta y = 0.05$

Plot and compare the computed solutions at grid points with the exact solution. Give comments on the benefit of using different interval lengths Δx and Δy . Provide examples that can provide the benefit of using different interval lengths.

13. Use the central difference method to derive the algebraic equation corresponding to the Laplace's equation in the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad 0 \leq x \leq 1, 0 \leq y \leq 1$$

with the boundary conditions of

$$u(x, 0) = 0, \quad u(x, 1) = 1/(1+x)^2 + 1, \quad 0 \leq x \leq 1$$

$$u(0, y) = y/(1+y^2), \quad u(1, y) = y/(4+y^2), \quad 0 \leq y \leq 1$$

Develop a computer program for solving the problem by dividing the unit square domain into different intervals as follows

- (a) $\Delta x = \Delta y = 0.2$
- (b) $\Delta x = \Delta y = 0.1$
- (c) $\Delta x = \Delta y = 0.05$

Then, plot to compare the computed solutions with the exact solution of

$$u(x, y) = y / ((1+x)^2 + y^2)$$

14. Use the central difference method to derive the algebraic equation corresponding to the Poisson's equation in the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -1 \quad 0 \leq x \leq 1, 0 \leq y \leq 1$$

with the boundary conditions of $u(x, y) = 0$ along the four edges. Divide the unit square domain into 3 equal intervals in both x - and y -directions. Compare the computed solutions at grid points with the exact solution of

$$u(x, y) = \frac{16}{\pi^4} \sum_{j=1}^{\infty} \sum_{k=1}^{\infty} \frac{\sin(j\pi x) \sin(k\pi y)}{j^3 k^2 + j^2 k^3}$$

15. Use the central difference method to derive the algebraic equation corresponding to the Poisson's equation in the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2(x^2 + y^2) \quad 0 \leq x \leq 1, 0 \leq y \leq 1$$

with the boundary conditions of

$$\begin{aligned} u(x, 0) &= 1 - x^2, & u(x, 1) &= 2(1 - x^2), & 0 \leq x \leq 1 \\ u(0, y) &= 1 + y^2, & u(1, y) &= 0, & 0 \leq y \leq 1 \end{aligned}$$

Divide the unit square domain into 3 equal intervals in both x - and y -directions. Compare the computed solutions at grid points with the exact solution of

$$u(x, y) = (1 - x^2)(1 + y^2)$$

16. Apply the central difference method to derive the algebraic equation for solving the one-dimensional boundary value problem governed by the ordinary differential equation

$$\frac{d^2 u}{dx^2} + u = 0 \quad 0 \leq x \leq \pi/2$$

with the boundary conditions of $u(0) = u(\pi/2) = 1$. Determine the solutions at grid points by using two models with the interval lengths of: (a) $\Delta x = \pi/4$ and (b) $\Delta x = \pi/10$. Compare the solutions at grid points with the exact solution of

$$u(x) = \sin x + \cos x$$

17. Apply the central difference method to derive the algebraic equation for solving the one-dimensional boundary value problem governed by the ordinary differential equation

$$\frac{d^2u}{dx^2} + u = \sin x \quad 0 \leq x \leq \pi/2$$

with the boundary conditions of $u(0) = u(\pi/2) = 0$. Determine the solutions at grid points by using two models with the interval lengths of: (a) $\Delta x = \pi/4$ and (b) $\Delta x = \pi/10$. Compare the solutions at grid points with the exact solution of

$$u(x) = -\frac{x}{2} \cos x$$

18. The Helmholtz equation is in the form

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + a(x, y)u = f(x, y)$$

If $a = -2$ and the function

$$f(x, y) = xy[(x^2 - 7)(1 - y^2) + (1 - x^2)(y^2 - 7)]$$

in the domain of $0 \leq x \leq 1$ and $0 \leq y \leq 1$ with $u = 0$ along the four boundaries, then the exact solution is

$$u(x, y) = (x - x^3)(y - y^3)$$

Use the central difference method to derive the corresponding algebraic equation. Develop a computer program to solve for the solutions at grid points by using the intervals with

(a) $\Delta x = \Delta y = 0.25$

(b) $\Delta x = \Delta y = 0.1$

Plot to compare the computed solutions at grid points with the exact solution above.

19. A square insulator with the inner and outer surface temperatures of 100 and 0 degrees, respectively, is shown in Fig. P8.19. Due to symmetry of the temperature distribution, only the lower-left quarter of the insulator can be used for modeling.

The steady-state temperature distribution can be determined by solving the Laplace's Eq. (8.15). Apply the algebraic equation corresponding to the Laplace's equation to the grid points of the model as shown in the figure. Such application leads to a set of equations that can be solved for the temperatures at grid points. Plot the computed temperature distribution by using contour lines.

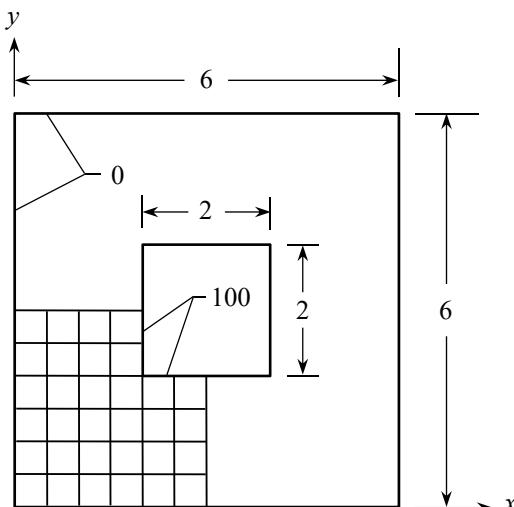


Figure P8.19.

20. Solve the parabolic equation in Example 8.3 again but by using only the left half of the bar because the temperature distribution is symmetric. Derive appropriate algebraic equation corresponding to the parabolic equation by using the explicit method. Then, develop a computer program to solve for the transient temperature response along the bar and compare the computed temperature at the 6 grid points with Table 8.3.
21. Employ the computer program developed in Problem 20 to solve the transient temperature response in the bar by using
- 26 grid points ($\Delta x = 0.02$)
 - 51 grid points ($\Delta x = 0.01$)
 - 101 grid points ($\Delta x = 0.005$)

Compare the solution obtained from each case with the exact solution. Give comments on the solution convergence as the model is refined.

22. Use the Crank-Nicolson method to solve the problem in Example 8.5 again but by modeling only the left half of the bar because its temperature distribution is symmetric. Develop a corresponding computer program to solve for the transient temperature response at the 6 grid points. Compare the computed solutions with those shown in Table 8.6.
23. The computer programs in Figs. 8.18, 8.21 and 8.23 are for the analysis of transient heat conduction in a bar by using the explicit, implicit and Crank-Nicolson methods, respectively. Employ these computer programs to study the convergence rates of the solutions obtained from the three methods by using the four models with the intervals of:

- $\Delta x = 0.02$
- $\Delta x = 0.01$
- $\Delta x = 0.005$
- $\Delta x = 0.001$

Note that the exact temperature at the middle of the bar at time $t = 0.20$ is 0.1389.

24. Use the explicit method to solve the parabolic equation

$$\frac{\partial u}{\partial t} - \frac{1}{\pi^2} \frac{\partial^2 u}{\partial x^2} = 0 \quad 0 \leq x \leq 1, t > 0$$

with the boundary conditions of

$$u(0, t) = u(1, t) = 0 \quad t > 0$$

and the initial condition of

$$u(x, 0) = \cos \pi(x - 0.5) \quad 0 \leq x \leq 1$$

Divide the domain into 4 equal intervals with 5 grid points. Use the time step $\Delta t = 0.2$ for $0 \leq t \leq 1$. Plot to compare the computed solution with the exact solution of $u(x, t) = e^{-t} \cos \pi(x - 0.5)$.

25. Solve Problem 24 again but by using a developed computer program. Divide the domain into:
 (a) 10 intervals, (b) 20 intervals and (c) 50 intervals. In each case, use the time step Δt about half of its critical time step. Plot to compare the solutions obtained with the exact solution. Give comments on the solution convergence to the exact solution.

26. Use the implicit method as explained in section 8.3.3 to solve the parabolic equation in Problem 24. Divide the domain into 4 equal intervals with 5 grid points. Use the time step of $\Delta t = 0.2$ for $0 \leq t \leq 1$. Set up a table to compare the computed solutions at grid points with the exact solution.
27. Use the algebraic equation derived in Problem 26 to develop a computer program. Then, employ the program to solve for the transient temperature response by dividing the domain into: (a) 10 intervals, (b) 20 intervals and (c) 50 intervals. Use appropriate time step Δt for the computation in each case. Give comments on the solution behaviors and their convergence to the exact solution.
28. Use the Crank-Nicolson method to establish the algebraic equation corresponding to the parabolic equation in Problem 24. Then, solve the problem by dividing the domain into 4 equal intervals with 5 grid points. Use the time step $\Delta t = 0.2$ for $0 \leq t \leq 1$. Compare the computed solutions at grid points with the exact solution.
29. Use the algebraic equation derived in Problem 28 to develop a computer program. Then, employ the program to solve the problem by dividing the domain into: (a) 10 intervals, (b) 20 intervals and (c) 50 intervals. Use appropriate time step Δt for each case and compared the computed solutions with the exact solution.
30. Derive the algebraic equation for solving the parabolic equation in the form

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 2 \quad 0 \leq x \leq 1, t > 0$$

with the boundary conditions of

$$u(0, t) = u(x, t) = 0 \quad t > 0$$

and the initial condition of

$$u(x, 0) = \sin \pi x + x(1-x) \quad 0 \leq x \leq 1$$

by using: (a) the explicit method and (b) the Crank-Nicolson method. Divide the domain into 4 equal intervals with 5 grid points and use the time step $\Delta t = 0.02$ for $0 \leq t \leq 0.1$. Set up a table to compare the computed solutions with the exact solution of

$$u(x, t) = e^{-\pi^2 t} \sin \pi x + x(1-x)$$

31. Use the algebraic equation derived in Problem 30 to develop a computer program. Employ the program to solve the problem by using: (a) the explicit method and (b) the Crank-Nicolson method. Divide the domain into 20 and 50 equal intervals and use the time step Δt about a half of the critical time for each case. Explain advantages and disadvantages of the two methods for solving the problem
32. Develop the algebraic equation for solving the parabolic equation in the form

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad 0 \leq x \leq 1, t > 0$$

with the boundary conditions of

$$u(0, t) = 1, \quad u(1, t) = 0, \quad t > 0$$

and the initial condition of

$$u(x, 0) = 1 - x - \frac{1}{\pi} \sin(2\pi x), \quad 0 \leq x \leq 1$$

by using: (a) the explicit method and (b) the Crank-Nicolson method. Divide the domain into 4 equal intervals with 5 grid points and use the time step $\Delta t = 0.01$ for $0 \leq t \leq 0.1$. Plot to compare the computed solutions with the exact solution of

$$u(x, t) = 1 - x - \frac{1}{\pi} e^{-4\pi^2 t} \sin(2\pi x)$$

- 33. Solve Problem 32 again but by developing a computer program. Divide the domain into 20 equal intervals and use appropriate time step Δt . Plot to compare the computed solutions with the exact solution.
- 34. Develop the algebraic equation for solving the parabolic equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}, \quad 0 \leq x \leq \pi, \quad t > 0$$

with the boundary conditions of

$$\frac{\partial u}{\partial x}(0, t) = \frac{\partial u}{\partial x}(\pi, t) = 0, \quad t > 0$$

and the initial condition of

$$u(x, 0) = \cos x, \quad 0 \leq x \leq \pi$$

by using: (a) the explicit method and (b) the Crank-Nicolson method. Divide the domain into 4 equal intervals with 5 grid points and use the time step $\Delta t = 0.05$ for $0 \leq t \leq 0.2$. Plot to compare the computed solutions with the exact solution of

$$u(x, t) = e^{-t} \cos x$$

- 35. Solve Problem 34 again but by developing a computer program. Divide the domain into 30 equal intervals and use appropriate time step Δt . Plot to compare the computed solutions with the exact solution.
- 36. Solve the hyperbolic equation representing the vibration of the string again by using the Courant numbers $C = 0.6, 0.8$ and 1.1 , respectively. Then, compare the computed solutions with the exact solution in Table 8.8.
- 37. Equation (8.75) that was derived for solving the string deflection after releasing it from the initial configuration is based on the condition of zero velocity. Re-derive the equation if the initial velocity is

$$\frac{\partial u}{\partial t}(x, t=0) = g(x)$$

where $g(x)$ is any given function.

38. Modify the computer program in Fig. 8.28 for the analysis of string vibration by dividing the string into 15 intervals with $\Delta x = 0.1$. Compare the computed solutions with the exact solution in Eq. (8.77). Plot the string deflections similar to those shown in Fig. 8.29 for $0 \leq t \leq 0.03$.
39. Derive the algebraic equation corresponding to the hyperbolic equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \quad 0 \leq x \leq 1, t \geq 0$$

with the boundary conditions of

$$u(0, t) = u(1, t) = 0 \quad t \geq 0$$

and the initial conditions of

$$u(x, 0) = \sin \pi x \quad 0 \leq x \leq 1$$

$$\frac{\partial u}{\partial t}(x, 0) = 0 \quad 0 \leq x \leq 1$$

Divide the domain into 4 equal intervals with 5 grid points and use the time step $\Delta t = 0.25$ for $0 \leq t \leq 1$. Set up a table to compare the computed solutions with the exact solution of

$$u(x, t) = \sin \pi x \cos \pi t$$

40. Use the algebraic equation derived in Problem 39 to develop a computer program. Then, employ the program to solve the problem by dividing the domain into: (a) 10 intervals, (b) 20 intervals and (c) 50 intervals. In each case, use appropriate time step Δt for determining the solution. Plot to compare the solution obtained from each model with the exact solution. Study the solutions and give comments on the solution convergence to the exact solution.
41. Derive the algebraic equation corresponding to the hyperbolic equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + 2e^{-t} \sin x \quad 0 \leq x \leq \pi, t \geq 0$$

with the boundary conditions of

$$u(0, t) = u(\pi, t) = 0 \quad t \geq 0$$

and the initial conditions of

$$u(x, 0) = \sin x \quad 0 \leq x \leq \pi$$

$$\frac{\partial u}{\partial t}(x, 0) = -\sin x \quad 0 \leq x \leq \pi$$

Divide the domain into 5 equal intervals with 6 grid points and use the time step $\Delta t = 0.2$ for $0 \leq t \leq 1$. Set up a table to compare the computed solutions with the exact solution of

$$u(x, t) = e^{-t} \sin x$$

42. Use the algebraic equation derived in Problem 41 to develop a computer program. Then, employ the program to solve the problem by dividing the domain into: (a) 10 intervals and (b) 20 intervals by using the time step of $\Delta t = 0.1$ and 0.05 , respectively. Plot to compare the solution obtained from each model with the exact solution. Compute and tabulate the solution errors at grid points for each case.
43. Derive the algebraic equation corresponding to the hyperbolic equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \quad 0 \leq x \leq 1, t \geq 0$$

with the boundary conditions of

$$u(0, t) = u(1, t) = 0 \quad t \geq 0$$

and the initial conditions of

$$u(x, 0) = \sin 2\pi x \quad 0 \leq x \leq 1$$

$$\frac{\partial u}{\partial t}(x, 0) = 2\pi \sin 2\pi x \quad 0 \leq x \leq 1$$

Divide the domain into 4 equal intervals with 5 grid points and use the time step $\Delta t = 0.25$ for $0 \leq t \leq 1$. Set up a table to compare the computed solutions with the exact solution of

$$u(x, t) = \sin 2\pi x (\sin 2\pi t + \cos 2\pi t)$$

44. Use the algebraic equation derived in Problem 43 to develop a computer program. Then, employ the program to solve the problem by dividing the domain into: (a) 10 intervals and (b) 20 intervals. Use appropriate time step Δt for each case. Compare the computed solution obtained from each model with the exact solution. Also plot to compare the solutions that vary with x -coordinate and time t with the exact solution.

Bibliography

- Atkinson, K. and Han, W., *Elementary Numerical Analysis*, Second Edition, John Wiley & Sons, New York, 2004.
- Bradie, B., *A Friendly Introduction to Numerical Analysis*, Pearson Education International, 2004.
- Buchanan, J. L. and Turner, P. R., *Numerical Methods and Analysis*, McGraw-Hill, New York, 1992.
- Burden, R. L. and Faires, J. D., *Numerical Analysis*, Fifth Edition, PWS Publishing, Boston, 2016.
- Chapman, S. J., *MATLAB Programming for Engineers*, Third Edition, Thomson International Edition, 2004.
- Chapra, S. C. and Canale, R. P., *Numerical Methods for Engineers*, Fifth Edition, McGraw-Hill International, 2006.
- Cheney, W. and Kincaid, D., *Numerical Mathematics and Computing*, Sixth Edition, Thomson International Edition, 2008.
- Dechaumphai, P., *Calculus and Differential Equations with MATLAB*, Alpha Science International, Oxford, 2016.
- Dechaumphai, P., *Calculus and Differential Equations with Mathematica*, Alpha Science International, Oxford, 2016.
- Fausett, L. V., *Applied Numerical Analysis Using MATLAB*, Prentice Hall, New Jersey, 1999.
- Ferziger, J. H., *Numerical Methods for Engineering Application*, Second Edition, John Wiley & Sons, New York, 1998.
- Gerald, C. F. and Wheatley, P. O., *Applied Numerical Analysis*, Seventh Edition, Pearson Education International, 2004.
- Gilat, A., *MATLAB: An Introduction with Applications*, Second Edition, John Wiley & Sons, New York, 2005.
- Gilat, A. and Subramaniam, V., *Numerical Methods for Engineers and Scientist: An Introduction with Applications Using MATLAB*, John Wiley & Sons, New York, 2007.
- Kaplan, W., *Advanced Calculus*, Fifth Edition, Addison-Wesley, Massachusetts, 2003.
- Lam, C. Y., *Applied Numerical Methods for Partial Differential Equations*, Prentice Hall, New York, 1994.
- Mathews, J. H. and Fink, K. D., *Numerical Methods Using MATLAB*, Fourth Edition, Pearson Education International, 2004.
- MATLAB Reference Guide*, The MathWorks, Inc, Massachusetts, 1992.
- Moore, H., *MATLAB for Engineers*, Second Edition, Pearson Education International, 2009.
- Nakamura, S., *Applied Numerical Methods with Software*, Prentice Hall International, 1991.
- Palm, W. J., *A Concise Introduction to MATLAB*, McGraw-Hill International, 2008.

Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T., *Numerical Recipes - The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1989.

Rao, S. S., *Applied Numerical Methods for Engineers and Scientists*, Pearson Education International, 2002.

Rice, R. J., *Numerical Methods, Software, and Analysis*, Second Edition, Academic Press, San Diego, 1993.

Ruskeepaa, H., *Mathematica Navigator: Mathematics, Statistics and Graphics*, Third Edition, Elsevier, San Diego, 2009.

Shewchuk, J. R., *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, School of Computer Science, Carnegie Mellon University, 1994.

Appendix A

Matrices

Understanding concepts of matrices and their uses are important in the study of the numerical methods. Matrices are used in the topics of solving a set of simultaneous equations, interpolation functions, least-square regressions and finite difference method. Concepts of matrices are employed in the development of the corresponding computer programs. In this appendix, definitions, properties and basic operations of matrices are presented.

A.1 Definitions

Matrices provide a shorthand scheme for dealing with systems of linear algebraic equations. For example, a set of three linear algebraic equations can be expressed in the form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \tag{A.1}$$

These equations can be written in matrix form as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \tag{A.2}$$

Or, in short

$$[A]\{X\} = \{B\} \tag{A.3}$$

where

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad (\text{A.4})$$

is a 3×3 matrix, i.e., having three rows and three columns. The coefficients a_{ij} , $i, j = 1, 2, 3$, in $[A]$ matrix are known. Similarly, the vector $\{B\}$ is a 3×1 matrix that contains the coefficients b_i , $i = 1, 2, 3$ as

$$\{B\} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix} \quad (\text{A.5})$$

In Eq. (A.3), the matrix $\{X\}$ is

$$\{X\} = \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} \quad (\text{A.6})$$

which has the same size as the matrix $\{B\}$ but consists of unknowns x_i , where $i = 1, 2, 3$.

For a practical problem, a set of algebraic equations in the form of Eq. (A.3) may be assembled from a large number of equations. If the set of algebraic equations consists of 1,000 equations, the matrix $[A]$ has 1,000 rows and 1,000 columns. The total number of coefficients in the matrix $[A]$ is 1,000,000 while the vectors $\{B\}$ and $\{X\}$ contain 1,000 coefficients and unknowns, respectively.

Often, the matrix $[A]$ is symmetric. In this case, it may be written shortly as

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \text{Symmetric} & & a_{33} \end{bmatrix} \quad (\text{A.7})$$

Furthermore, if the coefficients in the symmetric matrix $[A]$ in Eq. (a.7) are

$$a_{ij} = \begin{cases} 0 & \text{when } i \neq j \\ 1 & \text{when } i = j \end{cases}$$

the matrix $[A]$ becomes

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (\text{A.8})$$

which is called an identity matrix and normally denoted as $[I]$.

If a matrix has only one row, it is called a row matrix. For example

$$[D] = \begin{bmatrix} x & 2x^2 & \frac{x^3}{3} \end{bmatrix} \quad (\text{A.9})$$

Similarly, if a matrix has only one column, it is called a column matrix. The matrices $\{B\}$ and $\{X\}$ in Eqs. (A.5) and (A.6) are column matrices.

A.2 Matrix Addition and Subtraction

Matrices can be added or subtracted if they have the same numbers of rows and columns. For example, as matrices $[Q]$ and $[R]$ both have the size of 2×3 ,

$$\begin{bmatrix} P \\ (2 \times 3) \end{bmatrix} = \begin{bmatrix} Q \\ (2 \times 3) \end{bmatrix} + \begin{bmatrix} R \\ (2 \times 3) \end{bmatrix} \quad (\text{A.10})$$

Matrix addition yields the matrix $[P]$ with the same size of 2×3 . Coefficients of the matrices in Eq. (A.10) can be written in tensor form as

$$P_{ij} = Q_{ij} + R_{ij}$$

where $i = 1, 2$ and $j = 1, 2, 3$. Such matrix addition can be written for computer programming in Fortran language as

```
DO 10 I=1,2
DO 10 J=1,3
10 P(I,J) = Q(I,J) + R(I,J)
```

A.3 Matrix Multiplication

A scalar and a matrix can be multiplied directly. As an example,

$$\alpha \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} \alpha c_{11} & \alpha c_{12} \\ \alpha c_{21} & \alpha c_{22} \end{bmatrix} \quad (\text{A.11})$$

However, for multiplication of two matrices such as $[Q]$ and $[R]$ with sizes of $i \times j$ and $k \times l$, the number of columns (j) in matrix $[Q]$ must be equal to the number of rows (k) in matrix $[R]$, i.e.,

$$\begin{bmatrix} P \\ (i \times \ell) \end{bmatrix} = \begin{bmatrix} Q \\ (i \times j) \end{bmatrix} \begin{bmatrix} R \\ (j \times \ell) \end{bmatrix} \quad (\text{A.12})$$

This leads to the resulting matrix $[P]$ with the size of $i \times \ell$ as shown in Eq. (A.12). Coefficients in the matrices can be written in tensor form as

$$P_{i\ell} = \sum_{m=1}^j Q_{im} R_{m\ell}$$

As an example, if the matrix $[Q]$ has the size of (2×3) while the matrix $[R]$ has the size of (3×4) , the matrix $[P]$ will then have the size of (2×4) . Such matrix multiplication can be written for computer programming in Fortran language as

```

DO 10 I = 1,2
DO 10 L = 1,4
DO 10 J = 1,3
10 P(I,L) = P(I,L) + Q(I,J)*R(J,L)

```

It is noted that the positions of matrices during their multiplication lead to different results, i.e.,

$$[Q][R] \neq [R][Q]$$

Thus, pre- or post-multiplication of a matrix by another matrix must be performed carefully.

A.4 Matrix Transpose

If we have a matrix $[C]$ defined by

$$[C] = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \quad (\text{A.13})$$

Then its transpose is

$$[C]^T = \begin{bmatrix} c_{11} & c_{21} \\ c_{12} & c_{22} \end{bmatrix} \quad (\text{A.14})$$

i.e., coefficients in the rows and columns are interchanged so that row i of the matrix becomes column i in its transpose.

The following properties of matrix transpose are frequently used in the derivation of the finite element equations

$$([Q]+[R])^T = [Q]^T + [R]^T \quad (\text{A.15})$$

$$([Q][R])^T = [R]^T [Q]^T \quad (\text{A.16})$$

A.5 Matrix Inverse

If $[A]$ is a square and non-singular matrix, then

$$[A]^{-1}[A] = [I] \quad (\text{A.17})$$

where $[A]^{-1}$ is the inverse matrix of matrix $[A]$. For a small set of algebraic equations (A.3), the solution $\{X\}$ can be easily obtained with the inverse matrix of matrix $[A]$ as shown below

$$\begin{aligned} [A]^{-1}[A]\{X\} &= [A]^{-1}\{B\} \\ [I]\{X\} &= [A]^{-1}\{B\} \\ \{X\} &= [A]^{-1}\{B\} \end{aligned} \quad (\text{A.18})$$

However, this procedure is not used to solve the unknown $\{X\}$ in practical problems with a large set of algebraic equations. This is mainly because the determination of $[A]^{-1}$ consumes a large computational time and computer memory as compared to other solution methods.

A.6 Matrix Partitioning

Matrix partitioning can simplify the application of boundary conditions on the set of algebraic equations. For example, a set of four algebraic equations is considered,

$$\left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array} \right] \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{Bmatrix} \quad (\text{A.19})$$

If x_1, x_2, b_3, b_4 are known quantities, Eq. (A.19) can be written in a form of sub-matrices as follow

$$\left[\begin{array}{c|c} [A_{11}] & [A_{12}] \\ \hline (2 \times 2) & (2 \times 2) \\ \hline [A_{21}] & [A_{22}] \\ (2 \times 2) & (2 \times 2) \end{array} \right] \begin{Bmatrix} \{X_1\} \\ \{X_2\} \end{Bmatrix} = \begin{Bmatrix} \{B_1\} \\ \{B_2\} \end{Bmatrix} \quad (\text{A.20})$$

where the matrices $\{X_1\}$ and $\{B_2\}$ are known while the matrices $\{X_2\}$ and $\{B_1\}$ are unknown. The unknown matrix $\{X_2\}$ can be determined by using the lower set of equations in the system of Eq. (A.20) as follows

$$\begin{aligned} [A_{21}]\{X_1\} + [A_{22}]\{X_2\} &= \{B_2\} \\ [A_{22}]\{X_2\} &= \{B_2\} - [A_{21}]\{X_1\} \\ \{X_2\} &= [A_{22}]^{-1} \left(\{B_2\} - [A_{21}]\{X_1\} \right) \end{aligned} \quad (\text{A.21})$$

After obtaining the matrix $\{X_2\}$, the upper set of equations in the system of Eq. (A.20) is used to solve for the matrix $\{B_1\}$ as

$$\{B_1\} = [A_{11}]\{X_1\} + [A_{12}]\{X_2\} \quad (\text{A.22})$$

A.7 Calculus of Matrices

Coefficients in a matrix can be differentiated or integrated in a usual manner. As an example, if a matrix $[A]$ is

$$[A] = \begin{bmatrix} x & x^2 \\ 2x^2 & 3x \end{bmatrix} \quad (\text{A.23})$$

Then the derivative of the matrix $[A]$ with respect to x is

$$\frac{d}{dx}[A] = \begin{bmatrix} 1 & 2x \\ 4x & 3 \end{bmatrix} \quad (\text{A.24})$$

Similarly, the integration of the matrix $[A]$ from a lower limit 0 to an upper limit L is

$$\begin{aligned} \int_0^L [A] dx &= \begin{bmatrix} \frac{x^2}{2} & \frac{x^3}{3} \\ \frac{2x^3}{3} & \frac{3x^2}{2} \end{bmatrix}_0^L \\ &= \begin{bmatrix} \frac{L^2}{2} & \frac{L^3}{3} \\ \frac{2L^3}{3} & \frac{3L^2}{2} \end{bmatrix} \end{aligned} \quad (\text{A.25})$$

It is noted that coefficients in a matrix can be differentiated or integrated conveniently by using symbolic mathematical software such as MATLAB, Mathematica, Maxima, etc. These software packages also help manipulating algebraic expressions and convert them to several high-level computer languages ready for programming directly.

Appendix B

MATLAB Fundamentals

B.1 Introduction

MATLAB is a powerful software for solving scientific and engineering problems. The software contains a large number of mathematical functions and commands that can be used easily and conveniently. MATLAB provides many advantages as compared to many conventional high-level computer languages, such as Fortran, Pascal and C, commonly used for developing computer software. MATLAB has built-in graphic and visualization capability that can be used in a friendly, non-intimidating fashion.

MATLAB stands for *Matrix Laboratory* because the basic data that used in computation are from the elements in matrices. The software is widely used in colleges and universities especially in learning science and engineering courses. The software is also employed in design and development of new products in industry.

B.2 MATLAB Environment

When the MATLAB software is open, a window called “MATLAB desktop” appears as shown in Fig. B.1. The window consists of many sub-windows which are:

- a) Command Window
- b) Current Folder Window
- c) Workspace Window
- d) Editor Window
- e) Figure Window

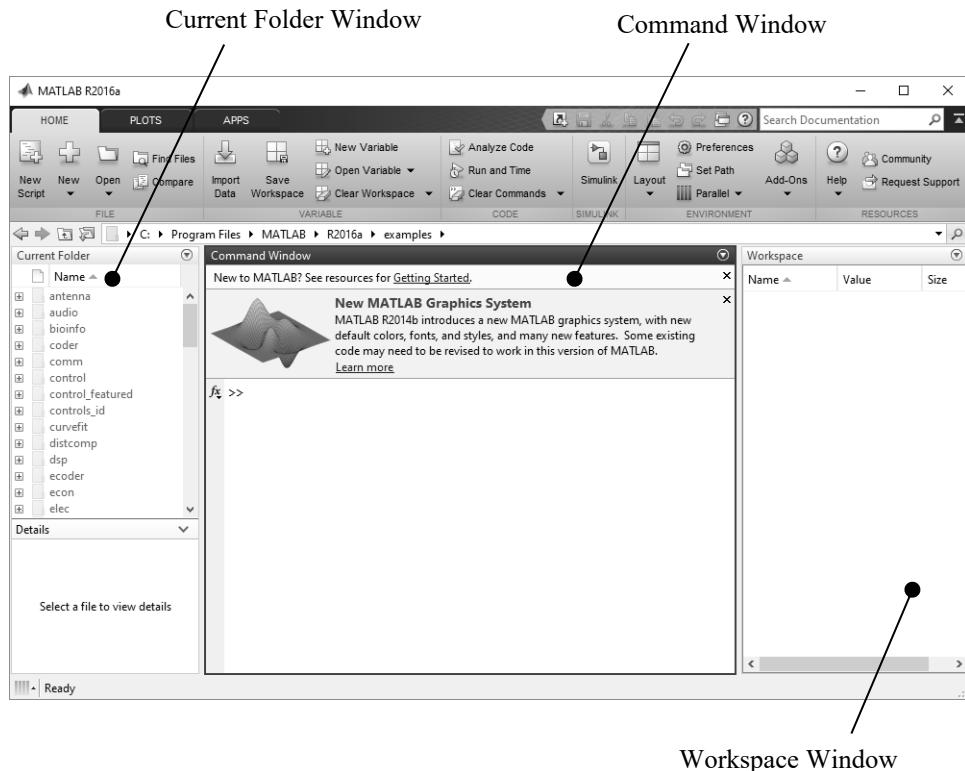


Figure B.1 MATLAB desktop and sub-windows.

B.2.1 Command Window

The Command Window is on the right-hand side of the MATLAB desktop as shown in Fig. B.1. In this window, users can directly type commands at the command prompt ($>>$). For example, the volume of a sphere with a radius of 3.5 can be determined by typing the following command,

```
>> volume = 4/3*pi*3.5^3
volume =
179.5944
```

After pressing the Enter key, MATLAB calculates the answer, saves it in a variable `volume` (array size 1x1) and displays the value of the variable on the screen as shown in Fig. B.2. If users do not want to show the result from the calculation on the screen, the semi-colon (`;`) must be added at the end of the command line before pressing the Enter key as follow,

```
>> area = 4*pi*3.5^2;
```

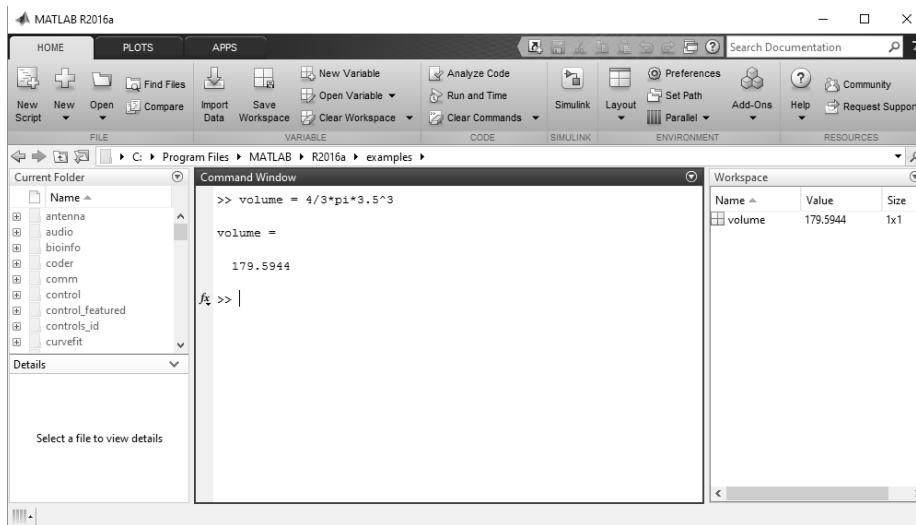


Figure B.2 Example of a command entered with its result displayed.

In this latter case, MATLAB stores the result from calculation in a variable called `area` and does not display the result on the screen as shown in Fig. B.3. It is noted that the value of π that was used in the above example is predefined by MATLAB with the variable named `pi`.

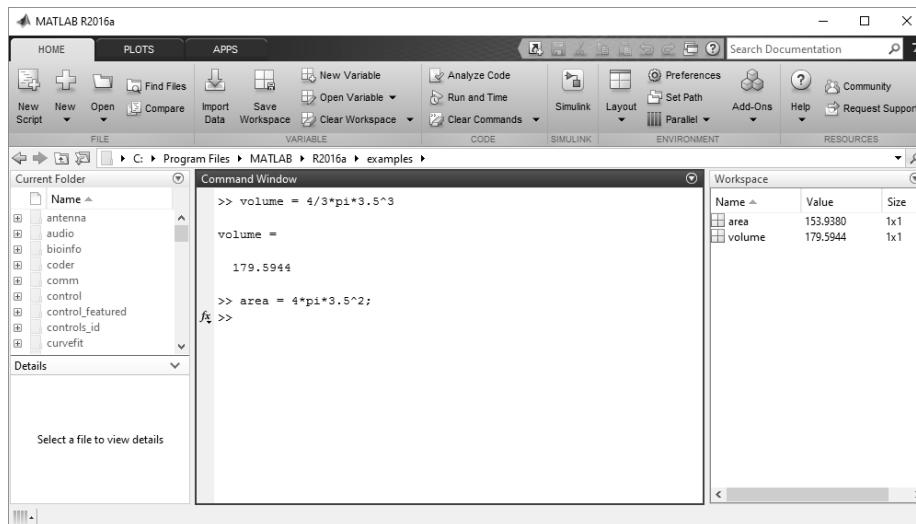


Figure B.3 Use of semi-colon at the end of command line to hide the displayed result.

For a set of commands, users can gather them in a single file. When that set of commands is needed, users can simply type the name of that file and press the Enter key. MATLAB will then execute all the commands in that file, line by line. The file that contains a set of commands is called the “script file”. The file is commonly known as the “m-file” because its extension is symbolized by “.m”.

B.2.2 Current Folder Window

The current folder window contains script files that are ready to use or recall for editing. All of the script files can be accessed via this window as highlighted in Fig. B.4.

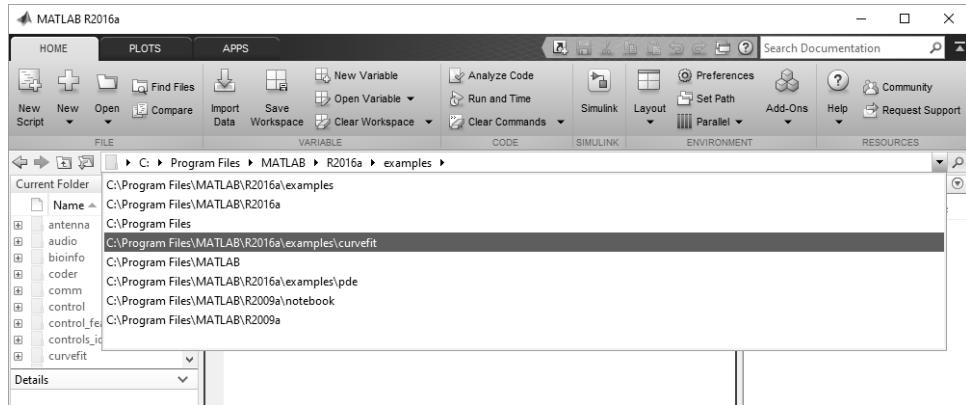


Figure B.4 Current Folder Window.

B.2.3 Editor Window

The Editor Window allows users to create or edit the existing m-file. An m-file can be created by selecting the New icon on the tool strip menu and clicking at the Script command.

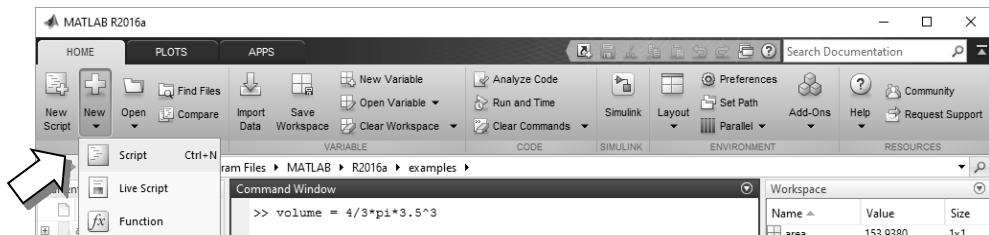


Figure B.5 Example for creating an m-file.

A new window then appears for creating a new script file as shown in Fig. B.6. The figure shows example of an m-file that was created in the Editor Window. The m-file calculates the volume of a sphere that has a radius of 3.5 and displays result on the screen. If this m-file is saved as “calc_volume.m”, the file can be executed by typing its name in the Command Window as follow,

```
>> calc_volume
The volume of the sphere =
179.5944
```

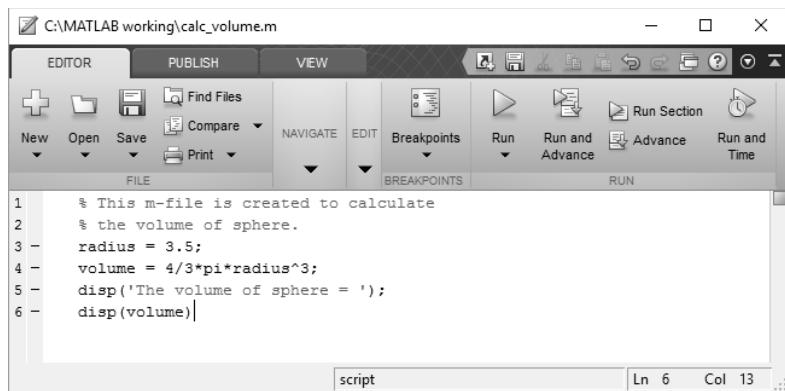


Figure B.6 Example of the `calc_volume.m` file.

B.2.4 Figure Window

The Figure Window displays different styles of graphics in two and three dimensions. As an example, a cosine function can be displayed by creating a file “`cos_x.m`” that contains commands as follows,

```

% cos_x.m: This file is created to calculate
%           and plot function of cos(x)
x = 0:0.1:8;
y = cos(x);
plot(x,y);

```

When the above m-file is executed, the Figure Window displays the plot of the function $\cos(x)$ as shown in Fig. B.7.

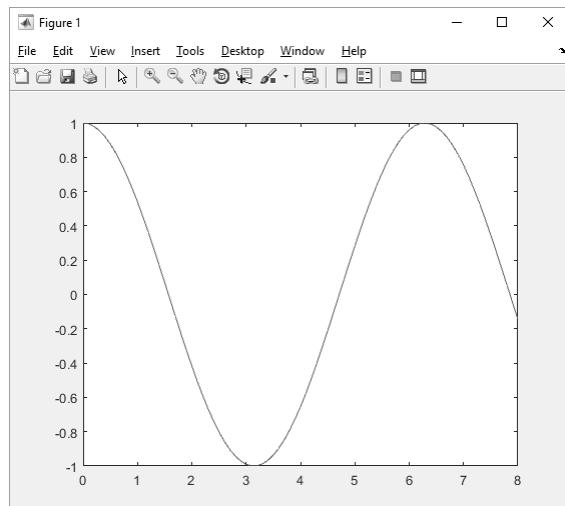
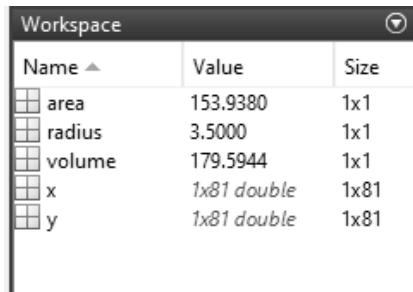


Figure B.7 Plotting the function $\cos(x)$ by using MATLAB.

B.2.5 Workspace Window

Variables are stored in of MATLAB memory called “workspace”. The workspace is displayed in Workspace Window on the top left corner of the desktop. An example of the workspace, after executing `calc_volume.m` and `cos_x.m`, is shown in Fig. B.8. The figure shows that the sizes of the variables `radius` and `volume` are 1×1 while the sizes for the variables `x` and `y` are 1×81 .



The screenshot shows the MATLAB workspace window titled "Workspace". It contains a table with three columns: "Name", "Value", and "Size". The variables listed are:

Name	Value	Size
area	153.9380	1x1
radius	3.5000	1x1
volume	179.5944	1x1
x	<i>1x81 double</i>	1x81
y	<i>1x81 double</i>	1x81

Figure B.8 Example of the Workspace Window.

Users can delete variables from the MATLAB memory by left-clicking on that variable and pressing the keyboard Delete button, or right-clicking on the variable and selecting the Delete command from the popup menu.

B.2.6 Help Window

Useful information is available in the Help Window. To open the window as shown in Fig. B.9, users may left-click on the icon  in the toolbar menu, or type words in the Command Window and press Enter.

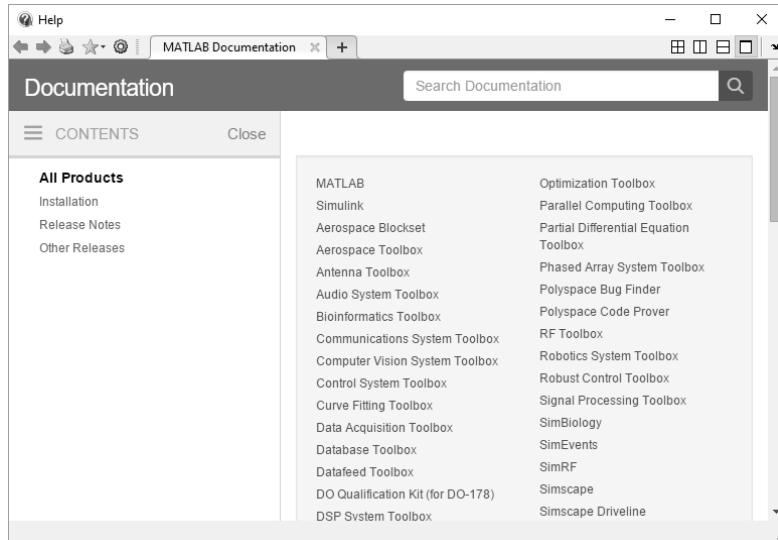


Figure B.9 Help Window.

B.3 Variables in MATLAB

Scalar and vector variables can be used in MATLAB. The use of these variables are explained below.

B.3.1 Scalar variable

The variable named ‘a’ for storing the value of 4 can be declared as,

```
>> a = 4  
a =  
    4
```

After pressing the Enter key, MATLAB displays value of the variable ‘a’. To assign values to many variables by using only one command line, comma (,) or semi-colon (;) is used between commands as,

```
>> a = 4; A = 6, x = 5;  
A =  
    6
```

From the above example, if the semi-colon (;) is used at the end of the command, MATLAB will not display the value of that variable. It is noted that the variable names are case-sensitive in MATLAB. As shown in the example above, MATLAB assigns the value of 4 to the lower-case letter variable ‘a’ and 6 to the upper-case letter variable ‘A’.

B.3.2 Vector variable

Values of variables are stored in the matrix form or array. For example, the scalar variable as shown in B.3.1 is stored in an array with size 1×1 . Array is a set of data normally stored in row and column form. If only one row or one column of an array is used to store a set of data, the array is called a vector. If both row and column of an array are used to store a set of data, the array is called a matrix. In this section, the arrays representing the vector and matrix are explained. To create a vector variable, users can simply enter the set of data in the square parentheses. For example,

```
>> a = [1 2 3 4 5]  
a =  
    1     2     3     4     5
```

The vector variable ‘a’ is then created to store the numbers 1 to 5 in a row. If users want to store these numbers in a column style, the semi-colon is used to separate the data as follow,

```
>> b = [1; 2; 3; 4; 5]  
b =  
    1  
    2  
    3  
    4  
    5
```

A row vector can be transposed to become a column vector by using the apostrophe (') symbol as,

```
>> b = [1 2 3 4 5]'  
b =  
1  
2  
3  
4  
5
```

A matrix, with size 3×3 for example, can be created by using the command,

```
>> c = [1 2 3; 4 5 6; 7 8 9]  
c =  
1 2 3  
4 5 6  
7 8 9
```

A data can be taken from the created vector or matrix. For example, the third data in the vector 'a', can be retrieved by typing,

```
>> a(3)  
ans =  
3
```

Similarly, the data from the third row and second column of the matrix 'c' can be retrieved by

```
>> c(3,2)  
ans =  
8
```

MATLAB contains built-in functions that can help users to create special matrices. For example, to create the null matrix with the size of 3×4 , the built-in function `zeros` may be used as,

```
>> d = zeros(3,4)  
d =  
0 0 0 0  
0 0 0 0  
0 0 0 0
```

The built-in function `ones` is used to create the unity matrix. For example,

```
>> e = ones(2,4)  
e =  
1 1 1 1  
1 1 1 1
```

B.3.3 Use of colon symbol

The colon (:) symbol between two numbers, while assigning values to a variable, leads to a set of data in arithmetic series from the first to the second number with the step size of one. For example,

```
>> s = 2:6
s =
    2     3     4     5     6
```

To create a set of data with a specific step size, the colon symbols are used between the three numbers. A set of data from the first to the third number with the step size equal to the second number is created as shown in the examples of the variables *t*, *u* and *v* below.

```
>> t = 2:0.5:4
t =
    2.0000    2.5000    3.0000    3.5000    4.0000

>> u = 4:-0.5:2
u =
    4.0000    3.5000    3.0000    2.5000    2.0000

>> v = 1:-0.75:-1.5
v =
    1.0000    0.2500   -0.5000   -1.2500
```

B.3.4 Displaying data

MATLAB normally displays the computed values with four decimal digits. For example,

```
>> 2.5 + 3.37
ans =
    5.8700
```

More decimal digits can be displayed by using the command,

```
>> format long
```

So that the computed value is displayed with fourteen decimal digits,

```
>> 2.5 + 3.37
ans =
    5.870000000000000
```

To display values with the original format again, the short format command is used,

```
>> format short
```

Another format that is normally used in calculation is the scientific format. To display a computed value in such format, the commands `format short e` and `format long e` may be used. For examples,

```

>> format short e
>> 2.5 + 3.37
ans =
    5.8700e+000

>> format long e
>> 2.5 + 3.37
ans =
    5.870000000000000e+000

```

B.3.5 Use of long commands

If a long command is not fitted within a single command line, users can use the ellipsis, three periods (...), at the end of the line before pressing the Enter key to pause execution. Users can then continue typing the rest of the command on the new line as shown in the example below.

```

>> fx = exp(-1.5/4)*...
(2-1.5)-1

fx =
-6.563553606045139e-001

```

B.4 Mathematical Operation

Mathematical operators in MATLAB are represented by the symbols + – * / and \wedge . Mathematical expressions are executed from left to right by following the order of precedence. Meanings of the symbols and order of precedence are explained in Tables B.1 and B.2.

Table B.1 Arithmetic Operations.

<i>Symbol</i>	<i>Operation</i>
\wedge	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction

The use of mathematical operators is explained by the following examples.

```

>> pi^2
ans =
    9.8696
>> x = 2*pi;
>> x/2.5
ans =
    2.5133

>> x = -3^2
x =
   -9

>> x = (-3)^2
x =
    9

```

Table B.2 Order of precedence.

<i>Precedence</i>	<i>Operations</i>
First	Parentheses, evaluated from innermost pair.
Second	Exponentiation, evaluated from left to right.
Third	Multiplication and division (same precedence) evaluated from left to right.
Fourth	Addition and subtraction (same precedence) evaluated from left to right.

These mathematical operators are used for matrix operations such as the scalar-matrix and matrix-matrix multiplications. For example,

```

>> a = [1 2 3];
>> b = [2; 4; 6];
>> a*b
ans =
    28

>> b*a
ans =
    2     4     6
    4     8    12
    6    12    18

```

The matrix c obtained earlier can be multiplied by itself using the command,

```

>> c*c
ans =
    30     34     42
    66     81     96
   102    126    150

```

or by using the exponentiation operator to obtain the same result as,

```
>> c^2
ans =
    30     34     42
    66     81     96
   102    126    150
```

To multiply matrix A by matrix B, the number of columns in matrix A must be equal to the number of rows in matrix B. If users try to multiply two matrices that have inappropriate sizes, such as multiplying matrix s (1×5) by matrix c (3×3), MATLAB will display the error message as follow,

```
>> s*c
??? Error using ==> *
Inner matrix dimensions must agree.
```

The mathematical operators can also be used for scalar-matrix operation. For example,

```
>> c*2
ans =
    2     4     6
    8    10    12
   14    16    18
```

To multiply element-by-element between two matrices, the period (.) must be used before the operator symbol. For example,

```
>> c.^2
ans =
    1     4     9
   16    25    36
   49    64    81
```

B.5 Built-in Functions

There are numerous built-in functions in MATLAB that can be used for scientific and engineering calculation as shown in Table B.3.

Table B.3 Some commonly used mathematical functions.

Function	Syntax
Natural logarithmic, $\ln(x)$	<code>log (x)</code>
Exponential, e^x	<code>exp (x)</code>
Square root, \sqrt{x}	<code>sqrt (x)</code>
Absolute value, $ x $	<code>abs (x)</code>

Function	Syntax
Sine function, $\sin(x)$	<code>sin(x)</code>
Cosine function, $\cos(x)$	<code>cos(x)</code>
Hyperbolic tangent, $\tanh(x)$	<code>tanh(x)</code>
Etc.	

MATLAB also contains built-in functions for rounding numbers, such as `round`, `ceil` and `floor`. These functions can be used by studying the following examples.

```
>> h = [-1.3 -1.6 1.4 1.7];
```

Function `round` is used to round numbers to the nearest integers as follow

```
>> round(h)
ans =
    -1      -2      1      2
```

Function `ceil` is used to round numbers to the nearest integers toward ∞ as follow

```
>> ceil(h)
ans =
    -1      -1      2      2
```

Function `floor` is used to round numbers to the nearest integers toward $-\infty$ as follow

```
>> floor(h)
ans =
    -2      -2      1      1
```

Some statistic built-in functions can be used conveniently as follows.

```
>> f = [3 5 4 7 6];
>> sum(f)
ans =
    25
>> min(f), max(f), mean(f), sort(f)
ans =
    3
ans =
    7
ans =
    5
ans =
    3     4     5     6     7
```

B.6 Plotting Graphs

MATLAB contains a number of functions for plotting graphs. For example, to plot a graph from the two sets of data,

```
>> x = [0:0.1:20];
>> y = 0.25.*x.*cos(x);
```

the function `plot` can be employed as,

```
>> plot(x,y)
```

MATLAB generates a plot with `x` values on the horizontal axis and the corresponding `y` values on the vertical axis as shown in Fig. B.10. Users can copy and use the plot for reports and presentation conveniently. Moreover, users can include additional information into the plot as shown in Fig. B.11, such as the name of the graph or the labels for `x` and `y` axes, by typing the following commands,

```
>> title('Plot of y versus x')
>> xlabel('Values of x')
>> ylabel('Values of y')
>> grid on
```

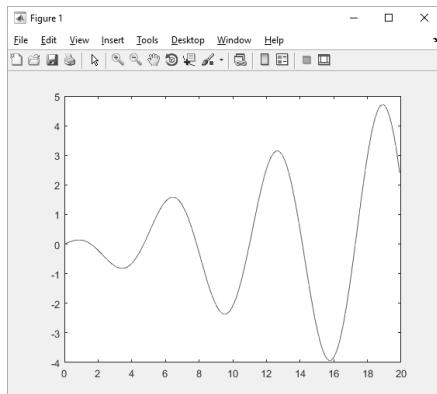


Figure B.10 Graph generated by the `plot` function.

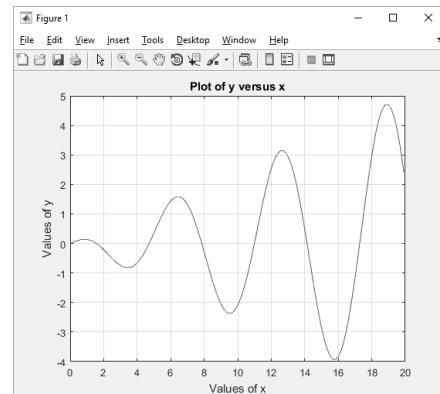


Figure B.11 Graph with additional information.

Normally, the graph created by using the function `plot` is displayed by continuous lines. Users can change the type or color of lines by adding symbols in function `plot`. For example, to plot a graph of vector `x` versus vector `y` with **red dashed line**, users may use the command,

```
>> plot(x,y,'--r')
```

where the symbol `(--)` in the function `plot` refers to a dashed line and the letter `(r)` indicates the red color.

Users can plot a graph and mark each data point with a data marker. For example, to plot a graph of vector x versus vector y with a continuous line and mark each data point with a small circle, the following command is used,

```
>> plot(x,y,x,y, 'o')
```

MATLAB will display the graph as shown in Fig. B.12. List of symbols for controlling the line types, colors and data makers are presented in Table B.4.

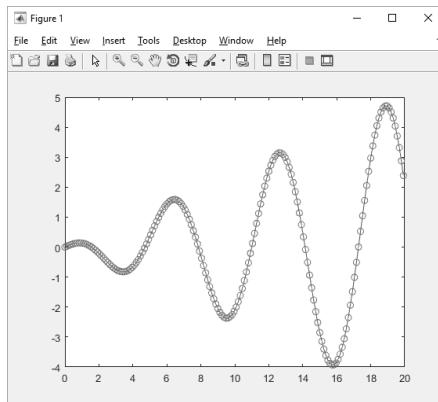


Figure B.12 Graph with data markers.

Table B.4 Symbols for line types, colors and data markers.

Line types		Data markers		Colors	
Continuous line	—	Dot	•	Black	k
Dotted line	:	Circle	o	Magenta	m
Dash-dotted line	-.	Cross	x	Blue	b
Dashed line	--	Plus sign	+	Cyan	c
		Triangle	^	Green	g
		Triangle (down)	v	Yellow	y
		Triangle (left)	<	Red	r
		Triangle (right)	>		
		Square	s		
		Diamond	d		

By using the function `plot`, the Figure Window is clear before executing the new command. To create new figures from a given array, users can use the function `subplot` with its syntax of,

```
subplot (u, v, w)
```

The command divides the Figure Window into an array with `u` rows and `v` columns. The variable `w` locates the position for displaying the output from the function `plot` when the function `subplot` is executed. For example, `subplot(2,2,3)` creates an array of four panes (two rows and two columns) and directs the next plot to the third pane (the lower-left corner). An example for using this function is as follows,

```
>> subplot(1,2,1); plot(x,y)
>> axis square
>> subplot(1,2,2); plot(x,y, 'o')
>> axis square
```

The corresponding plots are shown in Fig. B.13.

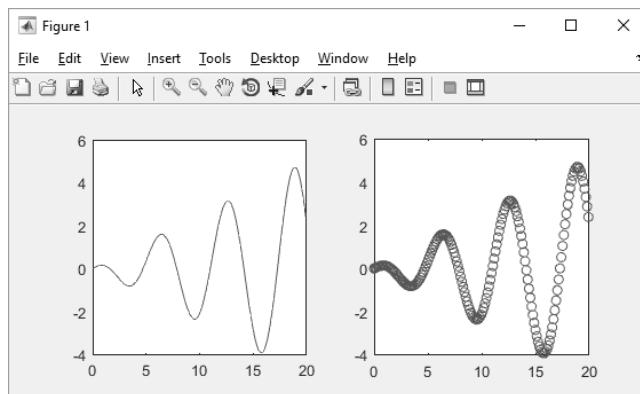


Figure B.13 Plots created by `subplot` command.

To plot a three-dimensional graph, users may use the function `plot3` which has the syntax of,

```
plot3(x, y, z)
```

For example, the following equations of x , y and z generate two three-dimensional curves that vary with t ,

$$\begin{array}{lll} \text{curve 1:} & x = (1 - t^2) \sin(t); & y = 1 + \cos(t); \\ \text{curve 2:} & x = \sin(t); & z = t \end{array}$$

By using the following commands, the plots are shown in Fig. B.14.

```
>> t = 0:pi/50:10*pi;
>> subplot(1,2,1); plot3((1-t.^2).*sin(t), 1+cos(t), t)
>> axis square
>> grid on
>> subplot(1,2,2); plot3(sin(t), cos(t), t)
>> axis square
>> grid on
```

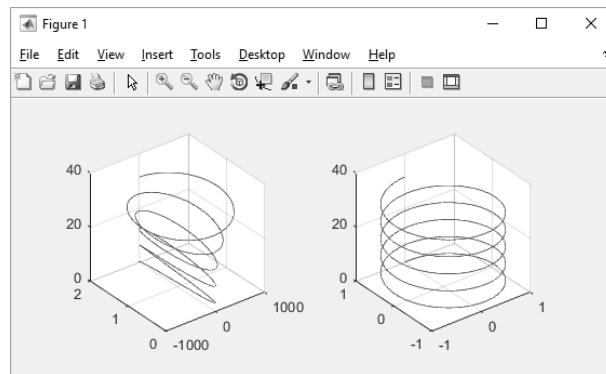


Figure B.14 Three-dimensional curves plotted by using the `plot3` function and `subplot` command.

Sometimes, users may need to plot a function with two variables, such as $z = f(x, y)$. The function represents a surface when plotted on x - y - z coordinates, such as,

$$f(x, y) = x(1-x)y(1-y)\tan^{-1}\left(100\left(\frac{x+y}{\sqrt{2}} - 0.8\right)\right)$$

where $0 \leq x \leq 1$ and $0 \leq y \leq 1$. To plot such surface, the boundaries in x - and y -directions are firstly defined,

```
>> x = 0:0.05:1;
>> y = 0:0.05:1;
```

Then, the grid points on x - y plane are generated by using the function `meshgrid`,

```
>> [X, Y] = meshgrid(x, y);
```

Next, the function $f(x, y)$ is evaluated at each grid point,

```
>> Z = X.* (1-X).*Y.* (1-Y).*atan(100*((X+Y)/sqrt(2)-0.8));
```

Finally, the function `mesh` is used to create the surface plot,

```
>> mesh(X, Y, Z)
>> view(248, 8.5) % control direction of viewpoint
```

The plot of the surface is shown in Fig.B.15 Another method to visualize the shape of a function is by using contour plot. A contour line in the contour plot represents a level or an elevation of the function. MATLAB uses the function `contour` to create the contour plots with the syntax of

```
contour(x, y, z, n)
```

where x , y and z are the matrix that can be prepared in the same way as those required by the `mesh` function and n is the number of contour lines.

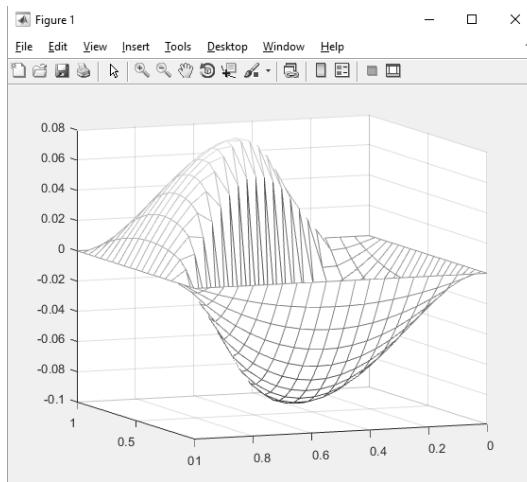


Figure B.15 A surface plot by using the `mesh` command.

Users can create the contour plot of the previous function with 20 contour lines by using the commands below. The resulting plot is shown in Fig. B.16.

```
>> contour(X,Y,Z,20)
>> axis square
```

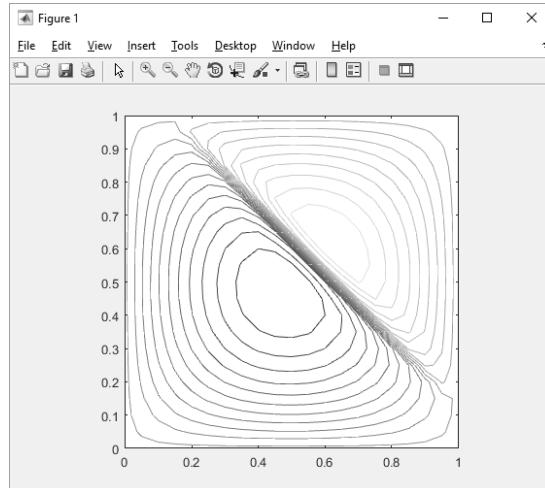


Figure B.16 A contour plot by using the `contour` command.

To create a contour plot with fill-in color, the `contourf` command may be used,

```
>> contourf(X,Y,Z,15)
>> axis square
```

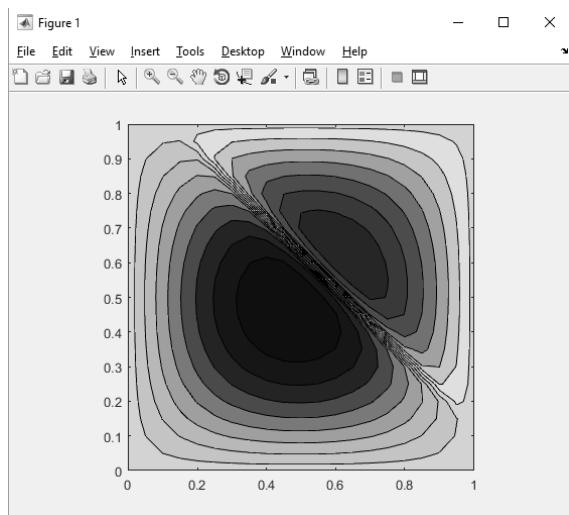


Figure B.17 A color contour plot by using the `contourf` command.

B.7 Programming

A set of command lines can be included in a file so that they can be executed later. The file that contains a set of command lines is called an ‘m-file’. Two types of the m-file presented herein are the Script and Function files.

B.7.1 Script file

Script file is a file that contains a set of command lines. By executing this file, it is equivalent to typing each line, one at a time, for execution. For example, a script file `calvel.m` that contains the two commands below can be created by selecting File>New>M-file on the window menu,

```
g = 9.81; m = 60.2; t = 12; Cd = 0.25;
v = sqrt(g*m/Cd)*tanh(sqrt(g*Cd/m)*t)
```

After saving it, the file can be executed by typing its name at the Command Window as,

```
>> calvel
v =
47.8435
```

MATLAB executes the commands in the file line-by-line. First, the program assigns values to the variable `g`, `m`, `t` and `Cd`. The program then calculates the values of variable `v` and displays them on the screen. All values used in script file are stored in the memory. Users can verify any value, such as that value of `Cd` (created by script file), by typing its name at the command prompt and press the Enter key as,

```
>> Cd
Cd =
0.25
```

B.7.2 Function file

Another type of the m-file is the function file. Unlike the script file, MATLAB deletes values of all variables created by the function file after execution. The function file is useful when the set of commands is executed repeatedly.

The first line of a function file must begin with the `function` command and a list of input and output variables. The structure of function file is as follows:

```
Function [output variable] = function_name(input variables)
% comment line
% other comment lines
...
Executable commands
...
(End of executable command)
```

As an example of a function file named as `fcalvel.m` is listed below,

```
function v = vel(m, t, Cd)
% Compute velocity from given data
% Input: mass(m), time(t), and drag coefficient(Cd)
% Output: velocity(v)
g = 9.81;
v = sqrt(g*m/Cd) *tanh(sqrt(g*Cd/m)*t);
```

To execute the function file above, the file's name and its input variables are typed. For example, to compute the variable `v(m/sec)` by using the variable `m = 60.2 kg`, `t = 12 sec` and `Cd = 0.25 kg/sec`, the following command is used,

```
>> fcalvel(60.2, 12, 0.25)
ans =
    47.8435
```

It is noted that, if users try to access the variable `g` created by the function file, the error message below is displayed.

```
>> g
??? Undefined function or variable 'g'.
```

B.7.3 Input and output commands

From the example of the script file above, if the variable `v` is to be determined by using the new value of `m`, users must open the script file and edit the value of variable before execute it. It will be better if the script file can wait for the users to input new data while the program is running. The `input` function can help users by displaying the message on the screen and waiting for users to enter new data from the keyboard. The input data will be stored in the specified variable. The syntax for the `input` function is,

```
variable = input ('message to interact with users')
```

For example, the command

```
t = input ('Time(t) : ')
```

is used for users to input the value of `Time(t)`. When the statement is executed, the message

```
Time(t) :
```

appears on the screen waiting for the users to type in the value. Users may prefer to store the input data as a character string. In this latter case, the second argument in the `input` function must be added as shown in the following syntax,

```
variable = input ('message to interact with users'), 's')
```

Another command that can help users to interact with the program is by using the `disp` function. The function displays text messages or values of variables on the screen. The syntax of this function is

```
disp (A)
```

where `A` is the variable name or the text message in a single quote. For example, the m-file below shows the use of different commands explained above.

```
function velocity
% Example of interactive function
% The function is created for
% calculating the velocity
g = 9.81;
m = input('Mass(kg) : ');
t = input('Time(sec) : ');
Cd = input('Drag coefficient (kg/m) : ');
disp(' ')
disp('Velocity (m/s):')
disp(sqrt(g*m/Cd)*tanh(sqrt(g*Cd/m)*t))
```

If the file is saved as `velocity.m`, users can execute the file by typing its name in the Command Window. After pressing the Enter key, the program will ask the users to enter the value of each variable before determining and displaying the value of velocity solution as follows,

```
>> velocity
Mass(kg) : 60.2
Time(sec) : 12
```

```
Drag coefficient (kg/m) : 0.25
```

```
Velocity (m/s) :
47.8435
```

To display the output data on the screen with a specific format, the `fprintf` function should be used instead of `disp` function. The syntax of this function is

`fprintf ('format', variable)`

where `format` contains the text to be printed on the screen and the special characters that describe format of the variable. To display the variable `v` which is equal 47.8435 on the screen with the floating point format in a field of eight characters wide, including three digits after the decimal point, the following command is used,

```
>> fprintf('The velocity is %8.3f m/s \n', v)
The velocity is    47.843 m/s
```

From the example of `fprintf` function, MATLAB displays the text inside the single quote until the program execution reaches the special character `%` or `\`. MATLAB then checks the meaning of that character and displays the variable according to the specified format. The special characters starting with the symbol `%` are called the conversion characters while the characters starting with `\` are called the escape characters. Examples of these special characters are described in Tables B.5 and B.6.

Table B.5. Some conversion characters for `fprintf` function.

Character	Description
<code>%d</code>	Display value as an integer
<code>%e</code>	Display value in exponential format
<code>%f</code>	Display value in floating point format
etc.	

Table B.6. Some escape characters for `fprintf` function.

Character	Description
<code>\n</code>	Skip to a new line
<code>\t</code>	Horizontal tab
etc.	

It is noted that when the program is executed to the escape character (\n) in the above example, MATLAB will move the command prompt to the new line and wait for the next command. To understand how the conversion characters affect the displaying format, the following statements are considered.

```
>> z = 7;
>> fprintf('The velocity is %18.4e m/s \n', z)
The velocity is      7.0000e+000 m/s
>> fprintf('The velocity is %8d m/s \n', z)
The velocity is      7 m/s
>> fprintf('The velocity is %8.4f m/s \n', z)
The velocity is    7.0000 m/s
```

Users can use the `fprintf` function to display multiple variables as,

```
>> a = 100; b = pi; c = 3*pi;
>> fprintf(' %5d %10.3f %8.5e \n ', a, b, c)
100      3.142 9.42478e+000
```

B.7.4 Read and write data file

To read or write data on a file, the file must be firstly identified. The file is open by using the `fopen` function. The syntax of this function is,

```
fid = fopen ('filename', 'permission')
```

where `fid` (short for file identification) is a number assigned to the file when it is opened. The `filename` is the name of the file including its extension and `permission` is the character for specifying the mode for opening file. For example, if the permission character is `r`, MATLAB will open the existing file for reading only. Example of the permission characters are shown in Table B.7.

Table B.7. Permissions used in `fopen` function.

File permission	Description
<code>r</code>	Open existing file for reading only
<code>r+</code>	Open existing file for reading and writing
<code>w</code>	Delete all data in existing file (or create a new file) and allow for writing only
<code>w+</code>	Delete all data in existing file (or create a new file) and allow for reading and writing.
etc.	

In addition, users can close the opened file by using the `fclose` function with the syntax of

```
fclose (fid)
```

After the file is opened, the data in that file can be read. MATLAB uses the function `fscanf` to read formatted data from the open file with the syntax of

```
variable = fscanf (fid, 'format', size)
```

where `fid` must be the same file identification of the file open, `format` is the pattern of the data, and `size` specifies the total number of data to be read from the file. There are three types of this argument:

- `n` Read data for `n` values and store them in a column vector.
- `[n m]` Read data for $n \times m$ values and store them in a matrix size $n \times m$.
- `Inf` Read data until reaching the end of the file.

In the proceeding section, the method for writing the formatted data on the screen by using `fprintf` function was explained. Users can also use the function to write formatted data into a file by first opening the file and using `fprintf` function with the syntax,

```
fprintf (fid, 'format', variable)
```

To understand how to use `fopen`, `fclose`, `fscanf` and `fprintf` function clearly, the following script file is considered.

```
% EXAMPLE PROGRAM
% OPEN INPUT FILE AND READ DATA
fid = fopen('input.dat', 'r');
neq = fscanf(fid, '%f', 1);
a = fscanf(fid, '%f', [neq neq]);
a = a.';
fclose(fid);
% OPEN OUTPUT FILE AND WRITE DATA
fid = fopen('output.dat', 'w');
fprintf(fid, '\n THIS TEXT WILL BE WRITTEN TO OUTPUT.DAT \n');
fprintf(    '\n THIS TEXT WILL BE WRITTEN ON SCREEN      \n');
fprintf(fid, ' %14.6e %14.6e %14.6e \n', a(2,1), a(2,2), a(2,3));
fprintf(    ' %14.6e %14.6e %14.6e \n', a(2,1), a(2,2), a(2,3));
fclose(fid);
```

The `input.dat` file contains the data,

```
3.
4.   -4.    0.
-1.   4.   -2.
0.   -2.    4.
```

When the script file above is executed, MATLAB opens the file `input.dat`, reads the first and only one data which is the number 3 and stores it in the variable `neq` which has its size defined as one. The

program then reads the rest of the data from the input file and stores them into the variable `a` which is a matrix with the size of $(n_{eq} \times n_{eq})$. To read the data in matrix form, MATLAB reads the first row from left to right and store them into the first column of variable `a`. After reading the data in such fashion, users must transpose the matrix so that it is in the same form as the input file. The program closes the `input.dat` file and opens another file for writing output data. The output file name is `output.dat`. The program then prints the text inside a single quote into the output file and on the screen by using the `fprintf` function. Next, the program prints values of the matrix `a` into the output file and on the screen. Finally, the program closes the output file and stop executing the script file. The `output.dat` file generated from the program contains the information as shown below,

```
THIS TEXT WILL BE WRITTEN TO OUTPUT.DAT
-1.000000e+000    4.000000e+000   -2.000000e+000
```

while the messages on the screen are,

```
THIS TEXT WILL BE WRITTEN ON SCREEN
-1.000000e+000    4.000000e+000   -2.000000e+000
```

B.7.5 Programming commands

Like other computer programs written in high-level languages, commands inside a MATLAB program are executed line-by-line from top to bottom. Executing series of commands, line by line, is not effective in solving scientific and engineering problems. Additional commands for making decision and performing iteration should be included to increase the efficiency of the computation.

B.7.5.1 Decision commands

To make a decision, the `if` command is used with its structure,

```
if logical expression
    statements
end
```

The command asks a question by using a logical expression for making decision. If the answer is true, the statements inside the `if` command is executed. Examples of the logical operators used in logical expressions are shown in Table. B.8.

Table B.8. Logical operators.

Logical operators	Example	Operation
<code>==</code>	<code>x == 4</code>	Equal to
<code>~=</code>	<code>x ~= 4</code>	Not equal to
<code><</code>	<code>y < x</code>	Less than
<code>></code>	<code>y > 5</code>	Greater than

Logical operators	Example	Operation
\leq	$10 \leq a/3$	Less than or equal to
\geq	$t \geq a$	Greater than or equal to
\sim	$\sim(t \geq a)$	Logical NOT
$\&$	$(x==4) \& (y>x)$	Logical AND
$ $	$(a < x) (x < y)$	Logical OR

To understand operations of the `if` command, the following function file is considered.

```
function grade(score)
% Determines whether score is good, fair or poor
% Input: numerical value of score (1-100)
% Output: display message (GOOD, FAIR or POOR)
if score >= 80
    disp('GOOD')
end
if score >= 50 & score < 80
    disp('FAIR')
end
if score < 50
    disp('POOR')
end
```

If the program is executed three times with the scores of 87, 64 and 30, the results are,

```
>> grade(87)
GOOD
>> grade(64)
FAIR
>> grade(30)
POOR
```

Another command for making decision is the `if...else` command with its structure of

```
if logical expression
    statements #1
else
    statements #2
end
```

From the `if...else` command above, the statements in #1 are executed when the logical expression is true while the statements in #2 are executed when the logical expression is false. For example, the example below shows the use of the `if...else` command.

```
function grade2(score)
% Determines whether score is pass or fail
% Input: numerical value of score (1-100)
% Output: display message (PASS or FAIL)
if score >= 50
    disp('PASS')
else
    disp('FAIL')
end
```

The results from executing the program are shown as below.

```
>> grade2(75)
PASS
>> grade2(49)
FAIL
```

Another form of the commands that can make a decision is the `if...elseif` command with its structure of,

```
If logical expression #1
    statements #1
elseif logical expression #2
    statements #2
elseif logical expression #3
    statements #3
    .
    .
    .
else
    statements
end
```

The command allows users to use multiple logical expressions for making complex decision. The example for using the `if...elseif` command is shown below.

```
function grade3(score)
% Determines whether score is good, fair or poor
% Input: numerical value of score (1-100)
% Output: display message (GOOD, FAIR or POOR)
if score >= 80
    disp('GOOD')
elseif score >= 50 & score < 80
    disp('FAIR')
else
    disp('POOR')
end
```

The results after executing the program are,

```
>> grade3(25)
POOR
>> grade3(89)
GOOD
>> grade3(57)
FAIR
```

B.7.5.2 Iteration commands

The two commands commonly used for making iteration are the `for` and `while` commands. The `while` command terminates the iteration by using a logical expression. The `for` command stops the iteration when the number of iterations is reached

- **for command**

The structure of the `for` command is

```
for index = first:increment:last
    statement 1
    .
    .
    .
    statement n
end
```

Operations of the `for` command (if `increment` value is positive) are as follows.

1. At the beginning of execution, a set of number which starts from the `first` to the `last` number with the step of `increment` is created. The `increment` value is set to one if it is not assigned.
2. The program then assigns the first number to the variable `index` and executes the statements inside the `for` command.
3. After all statements in the `for` command are executed, the program then checks the value of the `index` variable. If the value of the `index` variable is greater than or equal to the `last` number, the operation terminated. If it is not, the program assigns the next number created from step 1 to the variable `index`.
4. The program executes the statements inside the `for` command again by repeating step 3 until the `index` variable is equal to the `last` number.

To understand the operations of the `for` command more clearly, the following example is studied.

```
for i = 1:5
    t = i^2;
    disp(t)
end
```

In this case, the program creates a set of number which starts from 1 to 5. The loop index `i` is assigned as one and the statements inside the `for` command are executed. At this step, the value of the index `i` is not yet equal to 5, so the program assigns the new value of 2 to the index `i` and the operation is repeated. The iteration is performed until the loop index `i` is 5 which is equal the `last` value. The `for` command is terminated and the program continues to execute the statement after the `end` statement. If the increment value is negative as shown in the example below,

```
for j = 10:-2:1
    disp(j)
end
```

The `for` command is terminated when the loop index `j` is less than or equal the `last` number. The example displays the numbers of 10, 8, 6, 4 and 2 respectively on the screen.

- **while command**

The structure of the `while` command is,

```
while logical expression
    statement 1
    .
    .
    statement n
end
```

The `while` command keeps executing statements inside its loop as long as the logical expression is true. Thus, the number of iteration is not known in advance. An example for using the `while` loop is shown below.

```
t = 10;
while t > 0
    t = t - 3;
    disp(t)
end
```

The displayed results from the example are 7, 4, 1 and -2 respectively.

- **pause command**

The `pause` command causes the program to stop and wait for user to press any key before continuing. The syntax for this command is,

```
pause or pause(n)
```

The `pause` command halts execution temporarily while the `pause(n)` suspends the execution for `n` seconds before continuing. Examples for the two commands above are shown below.

```
for i = 1:3
    pause
    disp(i)
end
```

The program uses the `for` command to display the numbers 1 to 3 on the screen but these numbers are displayed after a key on the keyboard is pressed.

```
for i = 1:3
    pause(5)
    disp(i)
end
```

The program displays the number 1 and waits for 5 second before showing the number 2. The program then waits for another 5 seconds before displaying the number 3.

MATLAB commands and examples presented in this Appendix highlight the capability of the software for helping scientists and engineers to solve their problems more effectively. The MATLAB commands presented herein are fundamental and essential. Further details for using MATLAB can be found in textbooks listed in bibliography.

Appendix C

Derivation of Fourth-Order Runge-Kutta Formula

The Runge-Kutta method is widely used to solve the ordinary differential equation. The most popular form is the fourth-order Runge-Kutta formula as shown in Eq. (7.55), which is

$$y_{i+1} = y_i + \left[\frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \right] h \quad (\text{C.1})$$

where

$$k_1 = f(x_i, y_i) \quad (\text{C.2a})$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_1\right) \quad (\text{C.2b})$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}hk_2\right) \quad (\text{C.2c})$$

$$k_4 = f(x_i + h, y_i + hk_3) \quad (\text{C.2d})$$

The Runge-Kutta formulas provide accurate solutions to the ordinary differential equations because their coefficients are determined by matching the formulas with the Taylor series. Chapter 7 shows the derivation for the coefficients of the second-order Runge-Kutta formula which are derived by matching with the Taylor series containing the terms up to h^2 . The coefficients of the fourth-order Runge-Kutta formula are determined in the same way by using the Taylor series that contains the terms up to h^4 . The orders of the Runge-Kutta formulas are thus identified by the highest order of h for the terms in the Taylor series used for matching.

The coefficients for the terms in the fourth-order Runge-Kutta formula can be derived by first writing Eqs. (C.1) - (C.2) in the general form of Eq. (7.38) as,

$$y_{i+1} = y_i + (\alpha_1 k_1 + \alpha_2 k_2 + \alpha_3 k_3 + \alpha_4 k_4)h \quad (C.3)$$

where

$$k_1 = f(x_i, y_i) \quad (C.4a)$$

$$k_2 = f(x_i + \beta_1 h, y_i + \beta_2 h k_1) \quad (C.4b)$$

$$k_3 = f(x_i + \beta_3 h, y_i + \beta_4 h k_1 + \beta_5 h k_2) \quad (C.4c)$$

$$k_4 = f(x_i + \beta_6 h, y_i + \beta_7 h k_1 + \beta_8 h k_2 + \beta_9 h k_3) \quad (C.4d)$$

Equations (C.3) - (C.4) contains 13 unknowns of $\alpha_i, i = 1$ to 4 and $\beta_j, j = 1$ to 9. These coefficients are determined by matching the fourth-order Runge-Kutta formula in (C.3) - (C.4) with the Taylor series that contains 5 terms (up to h^4). The matching leads to only 11 equations. Thus, 2 unknown coefficients are assigned so that the remaining 11 coefficients can be determined from 11 equations.

By assigning the 2 unknown coefficients, the fourth-order Runge-Kutta formula can be written in many forms. The most popular form is shown by Eqs. (C.1) - (C.2). Derivation of the fourth-order Runge-Kutta formula in this form starts by writing Eqs. (C.1) - (C.2) as

$$y_{i+1} = y_i + (a k_1 + b k_2 + c k_3 + d k_4)h \quad (C.5)$$

where

$$k_1 = f(x_i, y_i) \quad (C.6a)$$

$$k_2 = f(x_i + mh, y_i + mh k_1) \quad (C.6b)$$

$$k_3 = f(x_i + nh, y_i + nh k_2) \quad (C.6c)$$

$$k_4 = f(x_i + ph, y_i + ph k_3) \quad (C.6d)$$

There are 7 unknown coefficients of a, b, c, d, m, n, p . In order to simplify the derivation, the following variables are defined,

$$f = f(x_i, y_i) \quad ; \quad f_x = \frac{\partial f}{\partial x} \quad ; \quad f_y = \frac{\partial f}{\partial y} \quad (C.7)$$

and

$$F_1 = f_x + f f_y \quad (C.8a)$$

$$F_2 = f_{xx} + 2f f_{xy} + f^2 f_{yy} \quad (C.8b)$$

$$F_3 = f_{xxx} + 3f f_{xxy} + 3f^2 f_{xyy} + f^3 f_{yyy} \quad (C.8c)$$

Since the Taylor series with the first 5 terms is,

$$y_{i+1} = y_i + f h + f' \frac{h^2}{2!} + f'' \frac{h^3}{3!} + f''' \frac{h^4}{4!} + \dots \quad (C.9)$$

By starting with $y' = f$, then,

$$\begin{aligned}
f' &= f_x + f_y y' = f_x + f_y f = F_1 \\
f'' &= f_{xx} + 2f f_{xy} + f^2 f_{yy} + f_y(f_x + f f_y) = F_2 + f_y F_1 \\
f''' &= f_{xxx} + 3f f_{xxy} + 3f^2 f_{xyy} + f^3 f_{yyy} + f_y(f_{xx} + 2f f_{xy} + f^2 f_{yy}) \\
&\quad + 3(f_x + f f_y)(f_{xy} + f f_{yy}) + f_y^2(f_x + f f_y) \\
&= F_3 + f_y F_2 + 3F_1(f_{xy} + f f_{yy}) + f_y^2 F_1
\end{aligned}$$

and the Taylor series in Eq. (C.9) becomes,

$$\begin{aligned}
y_{i+1} &= y_i + f h + \frac{1}{2} F_1 h^2 + \frac{1}{6} (F_2 + f_y F_1) h^3 \\
&\quad + \frac{1}{24} [F_3 + f_y F_2 + 3(f_{xy} + f f_{yy}) F_1 + f_y^2 F_1] h^4 + \dots
\end{aligned} \tag{C.10}$$

By applying the two-variable Taylor series expansion according to Eq. (7.45) to k_2, k_3, k_4 in Eq. (C.6),

$$\begin{aligned}
k_1 &= f \\
k_2 &= f + mhF_1 + \frac{1}{2}m^2h^2F_2 + \frac{1}{6}m^3h^3F_3 + \dots \\
k_3 &= f + nhF_1 + \frac{1}{2}h^2(n^2F_2 + 2mnf_yF_1) \\
&\quad + \frac{1}{6}h^3(n^3F_3 + 3m^2nf_yF_2 + 6mn^2(f_{xy} + f f_{yy})F_1) + \dots \\
k_4 &= f + phF_1 + \frac{1}{2}h^2(p^2F_2 + 2npf_yF_1) \\
&\quad + \frac{1}{6}h^3(p^3F_3 + 3n^2pf_yF_2 + 6np^2(f_{xy} + f f_{yy})F_1 + 6mnpf_y^2F_1) + \dots
\end{aligned}$$

Then, substituting k_1, k_2, k_3, k_4 into Eq. (C.5) to yield,

$$\begin{aligned}
y_{i+1} &= y_i + (a+b+c+d)hf + (bm+cn+dp)h^2F_1 \\
&\quad + \frac{1}{2}(bm^2+cn^2+dp^2)h^3F_2 + \frac{1}{6}(bm^3+cn^3+dp^3)h^4F_3 \\
&\quad + (cmn+dnp)h^3f_yF_1 + \frac{1}{2}(cm^2n+dn^2p)h^4f_yF_2 \\
&\quad + (cmn^2+dnp^2)h^4(f_{xy} + f f_{yy})F_1 + dmnpf_y^2F_1 + \dots
\end{aligned} \tag{C.11}$$

Finally, by matching the coefficients in Eqs. (C.10) and (C.11), a total of 8 nonlinear algebraic equations is obtained as,

$$\begin{aligned}
a+b+c+d &= 1 \\
bm+cn+dp &= 1/2 \\
bm^2+cn^2+dp^2 &= 1/3
\end{aligned}$$

$$\begin{aligned}
 bm^3 + cn^3 + dp^3 &= 1/4 \\
 cmn + dnp &= 1/6 \\
 cmn^2 + dnp^2 &= 1/8 \\
 cm^2n + dn^2p &= 1/12 \\
 dmnp &= 1/24
 \end{aligned}$$

leading to the coefficient values of,

$$\begin{aligned}
 a = d &= 1/6 & ; & & b = c &= 1/3 \\
 m = n &= 1/2 & ; & & p &= 1
 \end{aligned}$$

Thus, the fourth-order Runge-Kutta formula in Eqs. (C.5) - (C.6) becomes,

$$\begin{aligned}
 y_{i+1} &= y_i + \left[\frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \right]h \\
 k_1 &= f(x_i, y_i) \\
 k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h k_1\right) \\
 k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}h k_2\right) \\
 k_4 &= f(x_i + h, y_i + h k_3)
 \end{aligned}$$

which is the most popular form of the Runge-Kutta formula as shown in Eqs. (C.1) - (C.2).

Appendix D

Mathematica Commands

Mathematica is a widely used software especially for research and education. The software contains a large number of efficient commands that can help reducing difficulty in developing computer programs. The software also include symbolic mathematics capability that can reduce complexity in manipulating algebraic expressions. Solutions may displayed in the forms of symbolic mathematics and in numeric formats.

This appendix demonstrates the use of Mathematica commands for solving examples in this book. The input commands are in bold letters while the outputs are in regular letters. Readers are encouraged to study the use of different commands in these examples to appreciate the capability of the software. Note that the example numbers in this appendix are corresponded to those in the chapters.

Example 1.1 Use the numerical method to solve an initial value problem governed by the ordinary differential equation

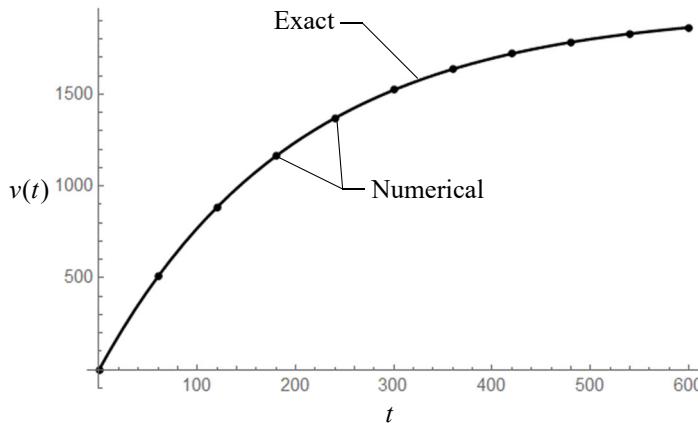
$$\frac{dv}{dt} + \frac{c}{m}v = g \quad 0 \leq t \leq 600$$

with the initial condition of $v(t=0)=0$.

```

m = 90000; c = 450; g = 9.8;
(* Numerical solution *)
numsol = NDSolve[{v'[t] + c/m v[t] == g, v[0] == 0}, v[t], {t, 0, 600}];
plot1 = Plot[v[t] /. numsol, {t, 0, 600}, PlotStyle -> Blue];
(* Exact solution *)
exact[t_] := (m g) / c (1 - E^(-c/m t));
exactsol = Table[{t, exact[t]}, {t, 0, 600, 60}];
plot2 = ListPlot[exactsol, PlotStyle -> Red];
Show[plot1, plot2]

```



Example 1.3 Mathematica can provide output in symbolic mathematics in addition to numeric formats.

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots$$

```
(* Sine function in form of infinite series *)
```

$$\sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

```
Sin[x]
```

```
(* Cosine function in form of infinite series *)
```

$$\sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

```
Cos[x]
```

Example 2.1 Finding root of equation

$$e^{-x/4} (2 - x) - 1 = 0$$

```
FindRoot[e^{-x/4} (2 - x) - 1 == 0, {x, 2}]
```

$$\{x \rightarrow 0.783596\}$$

Example 2.12 Finding roots of simultaneous equations

$$\begin{bmatrix} 1 & 2 & 1 & 4 \\ x_1 & 2x_1 & 0 & x_4^2 \\ x_1^2 & 0 & x_3 & 1 \\ 0 & 3 & 0 & x_3 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 20.700 \\ 15.880 \\ 21.218 \\ 21.100 \end{Bmatrix}$$

```
FindRoot[\{x_1 + 2 x_2 + x_3 + 4 x_4 == 20.700,
x_1 x_1 + 2 x_1 x_2 + x_4 x_4 == 15.880,
x_1^2 x_1 + x_3 x_3 + x_4 == 21.218,
3 x_2 + x_3 x_4 == 21.100\},
{\{x_1, 1}, {x_2, 1}, {x_3, 1}, {x_4, 1}}]
{x_1 \rightarrow 1.2, x_2 \rightarrow 5.6, x_3 \rightarrow 4.3, x_4 \rightarrow 1.}
```

Example 3.4 Solving a set of linear algebraic equations

$$\begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 400 \\ 400 \\ 400 \end{Bmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{pmatrix}; \mathbf{B} = \begin{pmatrix} 400 \\ 400 \\ 400 \end{pmatrix};$$

```
LinearSolve[A, B]
```

$$\{ \{ 450 \}, \{ 350 \}, \{ 275 \} \}$$

Example 3.11 Finding matrix inversion

$$[A]^{-1} = \begin{bmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}^{-1}$$

$$\mathbf{A} = \begin{pmatrix} 4 & -4 & 0 \\ -1 & 4 & -2 \\ 0 & -2 & 4 \end{pmatrix};$$

Inverse[A]

$$\left\{ \left\{ \frac{3}{8}, \frac{1}{2}, \frac{1}{4} \right\}, \left\{ \frac{1}{8}, \frac{1}{2}, \frac{1}{4} \right\}, \left\{ \frac{1}{16}, \frac{1}{4}, \frac{3}{8} \right\} \right\}$$

Example 3.15 Solving a set of linear algebraic equations when $[A]$ is symmetric.

$$\begin{bmatrix} 4 & 3 & 1 \\ 3 & 5 & 2 \\ 1 & 2 & 6 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} 3,125 \\ 3,650 \\ 2,800 \end{Bmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 4 & 3 & 1 \\ 3 & 5 & 2 \\ 1 & 2 & 6 \end{pmatrix}; \quad \mathbf{B} = \begin{pmatrix} 3125 \\ 3650 \\ 2800 \end{pmatrix};$$

LinearSolve[A, B]

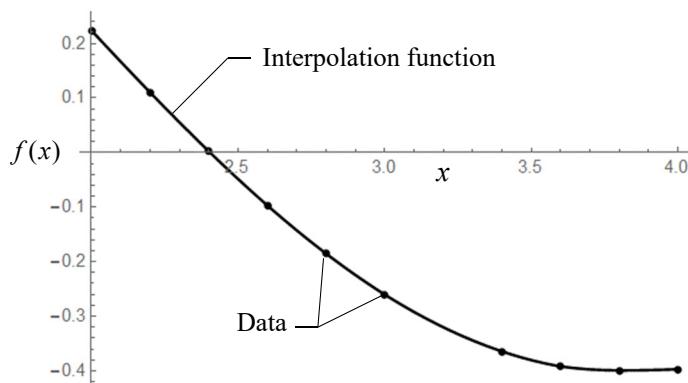
$$\{ \{ 450 \}, \{ 350 \}, \{ 275 \} \}$$

Example 4.3 Mathematica employs the Hermite method to provide interpolating solution with high accuracy.

```

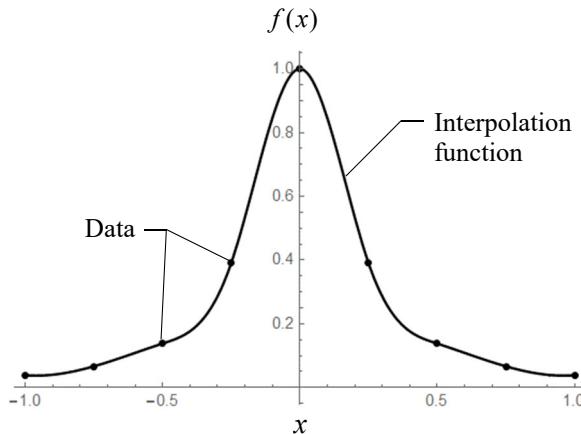
data = {{2.0, .2239}, {2.2, .1104}, {2.4, .0025}, {2.6, -.0968},
        {2.8, -.1850}, {3.0, -.2601}, {3.4, -.3643}, {3.6, -.3918},
        {3.8, -.3992}, {4.0, -.3971}};
func = Interpolation[data];
func[3.2]
Show[Plot[func[x], {x, 2.0, 4.0}], ListPlot[data, PlotStyle -> Red]]
-0.320133

```



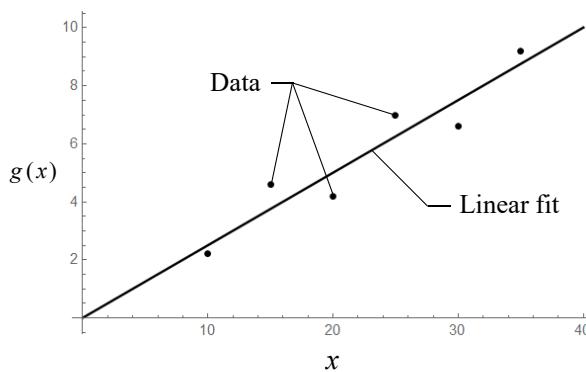
Example 4.8 Mathematica also includes the B-spline method which is slightly different from the interpolating method by using the cubic spline.

```
f[x_] := 1 / (1 + 25 x2);
data = {{-1.00, f[-1.00]}, {-0.75, f[-0.75]}, {-0.50, f[-0.50]},
{-0.25, f[-0.25]}, {0, f[0.00]}, {0.25, f[0.25]}, {0.50, f[0.50]},
{0.75, f[0.75]}, {1.00, f[1.00]}};
func = Interpolation[data, Method -> "Spline"];
Show[Plot[func[x], {x, -1, 1}], ListPlot[data, PlotStyle -> Red]]
```



Example 5.1 Linear regression.

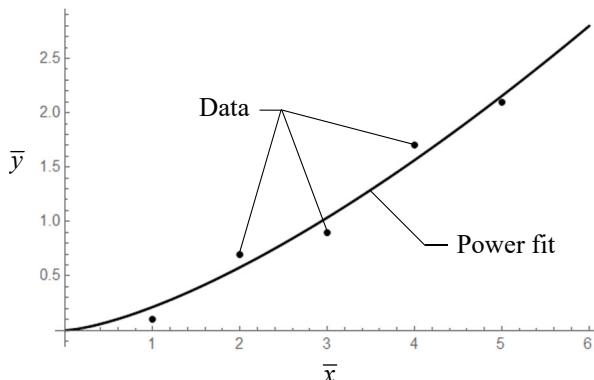
```
data = {{10, 2.2}, {15, 4.6}, {20, 4.2}, {25, 7.0}, {30, 6.6}, {35, 9.2}};
linear = Fit[data, {1, x}, x]
Show[Plot[linear, {x, 0, 40}], ListPlot[data, PlotStyle -> Red]]
0.00190476 + 0.250286 x
```



Example 5.3 Nonlinear regression.

```
data = {{1, 0.1}, {2, 0.7}, {3, 0.9}, {4, 1.7}, {5, 2.1}};
nonlinear = NonlinearModelFit[data, a x^b, {a, b}, x];
Normal[nonlinear]
Show[Plot[nonlinear[x], {x, 0, 6}], ListPlot[data, PlotStyle -> Red]]
```

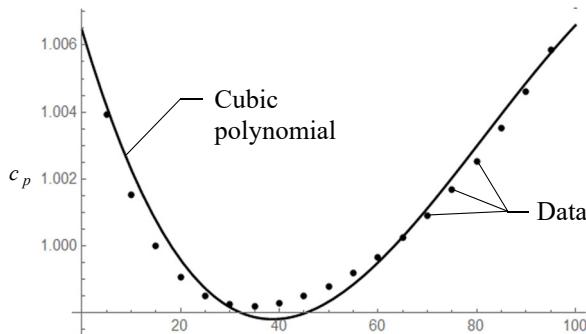
$0.214483 x^{1.43338}$



Example 5.4 Polynomial regression.

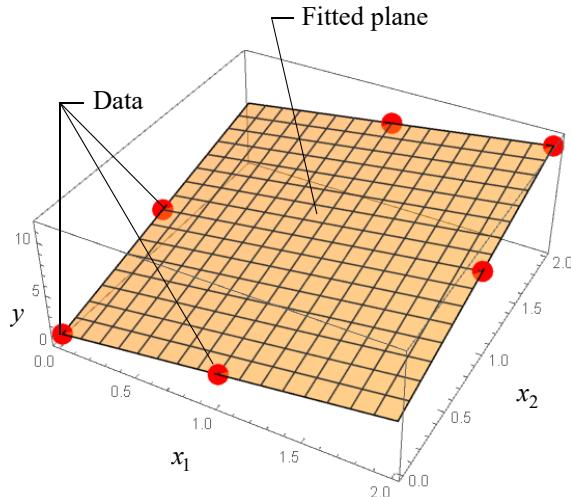
```
data = {{0, 1.00762}, {5, 1.00392}, {10, 1.00153}, {15, 1.00000},
{20, 0.99907}, {25, 0.99852}, {30, 0.99826}, {35, 0.99818},
{40, 0.99828}, {45, 0.99849}, {50, 0.99878}, {55, 0.99919},
{60, 0.99967}, {65, 1.00024}, {70, 1.00091}, {75, 1.00167},
{80, 1.00253}, {85, 1.00351}, {90, 1.00461}, {95, 1.00586},
{100, 1.00721}};
cubic = Fit[data, {1, x, x^2, x^3}, x]
Show[Plot[cubic, {x, 0, 100}], ListPlot[data, PlotStyle -> Red]]
```

$1.00645 - 0.000498571 x + 8.45381 \times 10^{-6} x^2 - 3.45339 \times 10^{-8} x^3$



Example 5.7 Multiple regression.

```
data = {{0, 0, 1}, {0, 1, 4}, {1, 0, 3}, {1, 2, 9}, {2, 1, 8}, {2, 2, 11}};
plane = Fit[data, {1, x, y}, {x, y}]
Show[Plot3D[plane, {x, 0, 2}, {y, 0, 2}, PlotRange -> {0, 12},
PlotStyle -> Opacity[.5]], Graphics3D[{Red, PointSize[0.04],
Map[Point, data]}]]
1. + 2. x + 3. y
```



Example 6.1 Integration.

$$I = \int_0^2 f(x) dx = \int_0^2 (2x^3 - 5x^2 + 3x + 1) dx$$

$$fx = 2x^3 - 5x^2 + 3x + 1;$$

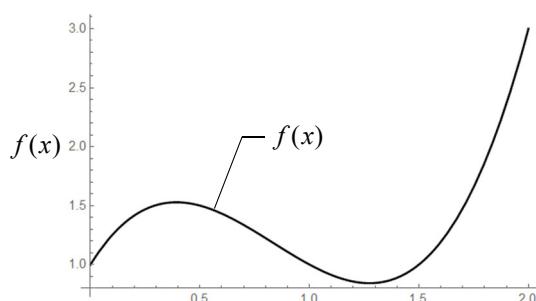
```
Plot[fx, {x, 0, 2}]
```

$$\int_0^2 fx dx$$

$$N\left[\int_0^2 fx dx, 20\right]$$

$$\frac{8}{3}$$

$$2.66666666666666666667$$



Example 6.1 Double integration.

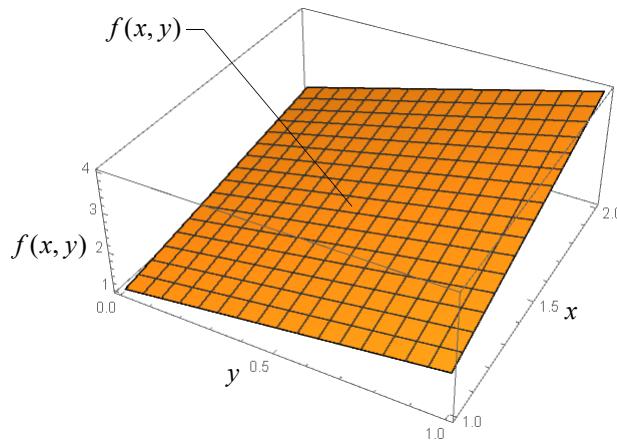
$$I = \int_0^1 \int_1^2 f(x, y) dx dy = \int_0^1 \int_1^2 (1+x)y dx dy$$

```
fxxy = (1 + x) y;
```

```
Plot3D[fxxy, {x, 0, 1}, {y, 1, 2}]
```

$$\int_0^1 \int_1^2 (1+x) y dx dy$$

$$N \left[\int_0^1 \int_1^2 (1+x) y dx dy \right]$$



$$\frac{5}{4}$$

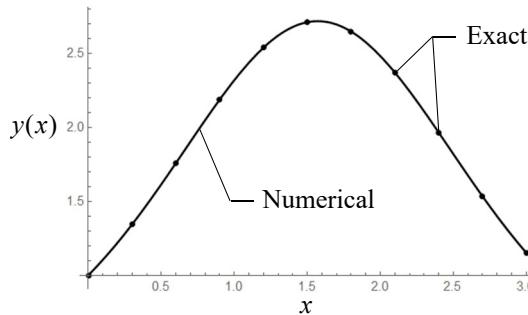
$$1.25$$

Example 7.1 Solving first-order ordinary differential equation.

$$\frac{dy}{dx} = y \cos x \quad 0 \leq x \leq 3$$

with the initial condition of $y(0) = 1$.

```
(* Numerical solution *)
numsol = NDSolve[{y'[x] - y[x] Cos[x] == 0,
    y[0] == 1}, y[x], {x, 0, 3}];
plot1 = Plot[y[x] /. numsol, {x, 0, 3}, PlotStyle -> Blue];
(* Exact solution *)
exact[x_] := e^Sin[x];
exactsol = Table[{x, exact[x]}, {x, 0, 3, .3}];
plot2 = ListPlot[exactsol, PlotStyle -> Red];
Show[plot1, plot2]
```

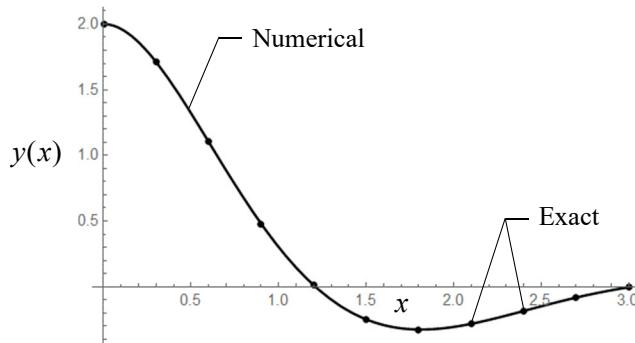


Example 7.6 Solving second-order ordinary differential equation.

$$\frac{d^2y}{dx^2} + 2 \frac{dy}{dx} + 4y = 0 \quad 0 \leq x \leq 3$$

with the initial conditions of $y(0) = 2$ and $dy/dx(0) = 0$.

```
(* Numerical solution *)
numsol = NDSolve[{y''[x] + 2 y'[x] + 4 y[x] == 0,
    y[0] == 2, y'[0] == 0}, y[x], {x, 0, 3}];
plot1 = Plot[y[x] /. numsol, {x, 0, 3}, PlotStyle -> Blue];
(* Exact solution *)
exact[x_] := 2 e^-x (Cos[\sqrt{3} x] + 1/\sqrt{3} Sin[\sqrt{3} x]);
exactsol = Table[{x, exact[x]}, {x, 0, 3, .3}];
plot2 = ListPlot[exactsol, PlotStyle -> Red];
Show[plot1, plot2]
```



Example 8.1 Solving an elliptic partial differential equation

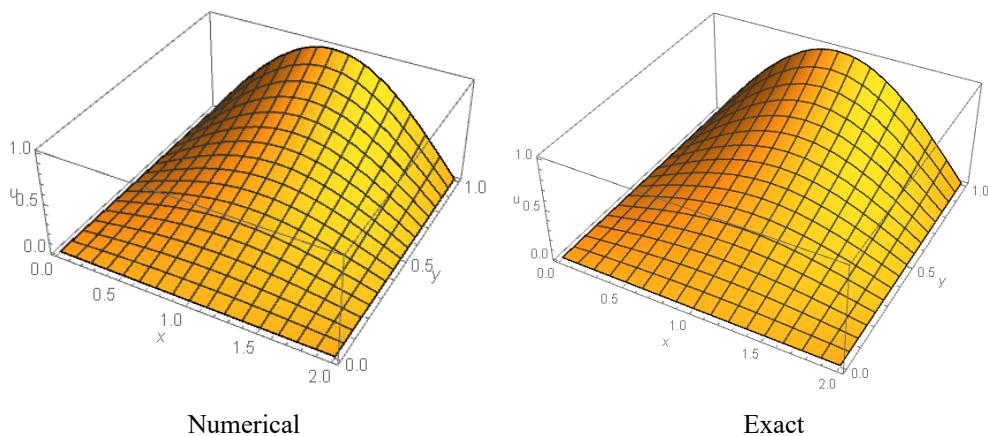
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad 0 \leq x \leq 2, \quad 0 \leq y \leq 1$$

with the boundary conditions of $u = 0$ along the left, right and bottom edges, while $u = \sin(\pi x/2)$ along the top edge.

```

 $a_0 = 0; a_1 = \text{Sin}\left[\frac{\pi x}{2}\right]; b_0 = 0; b_1 = 0;$ 
 $x_0 = 0; x_1 = 2; y_0 = 0; y_1 = 1; n_x = 20; n_y = 10;$ 
 $h_x = N[(x_1 - x_0) / n_x]; h_y = N[(y_1 - y_0) / n_y];$ 
Do[ $\xi_i = x_0 + i h_x, \{i, 0, n_x\}$ ];
Do[ $\eta_j = y_0 + j h_y, \{j, 0, n_y\}$ ];
vars = Table[ui,j, {i, 0, nx}, {j, 0, ny}];
bound1 = Table[{ui,0 == a0 /. x → ξi, ui,ny == a1 /. x → ξi}, {i, 0, nx}];
bound2 = Table[{u0,j == b0 /. y → ηj, unx,j == b1 /. y → ηj}, {j, ny - 1}];
eqns = Table[ $\frac{u_{i+1,j} - 2 u_{i,j} + u_{i-1,j}}{h_x^2} + \frac{u_{i,j+1} - 2 u_{i,j} + u_{i,j-1}}{h_y^2} = 0$ 
 /. {x → ξi, y → ηj}, {i, nx - 1}, {j, ny - 1}];
sol = vars /. Solve[Flatten[{eqns, bound1, bound2}],
 Flatten[vars]][[1]];
uappr = ListInterpolation[sol, {{x0, x1}, {y0, y1}}];
plotapprox = Plot3D[uappr[x, y], {x, 0, 2}, {y, 0, 1},
 AxesLabel → {x, y, "u"}, Mesh → 15];
(* Exact solution *)
exact =  $\frac{\text{Sin}\left[\frac{\pi x}{2}\right] \text{Sinh}\left[\frac{\pi y}{2}\right]}{\text{Sinh}\left[\frac{\pi}{2}\right]}$ ;
plotexact = Plot3D[exact, {x, 0, 2}, {y, 0, 1},
 AxesLabel → {x, y, "u"}, Mesh → 15];
GraphicsGrid[{{plotapprox, plotexact}}, ImageSize → Large]

```

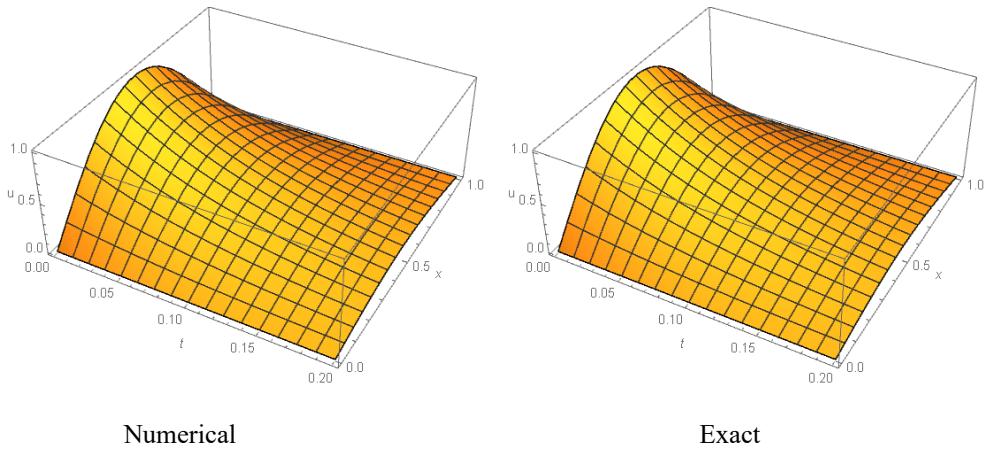


Example 8.4 Solving a parabolic partial differential equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad 0 \leq x \leq 1, \quad 0 \leq t \leq 0.2$$

with the boundary conditions of $u(0,t) = u(1,t) = 0$ and the initial condition of $u(x,0) = \sin(\pi x)$.

```
pde = D[u[x, t], t] == D[u[x, t], {x, 2}];
ic = {u[0, t] == 0, u[1, t] == 0, u[x, 0] == Sin[\[Pi] x]};
sol = NDSolve[{pde, ic}, u[x, t], {x, 0, 1}, {t, 0, .2}];
plotapprox = Plot3D[u[x, t] /. sol, {t, 0, 0.2}, {x, 0, 1},
  AxesLabel \[Rule] {t, x, "u"}, Mesh \[Rule] 15];
(* Exact solution *)
uexact[x_, t_] = E^{-\pi^2 t} Sin[\[Pi] x];
plotexact = Plot3D[uexact[x, t], {t, 0, 0.2}, {x, 0, 1},
  AxesLabel \[Rule] {t, x, "u"}, Mesh \[Rule] 15];
GraphicsGrid[{{plotapprox, plotexact}}, ImageSize \[Rule] Large]
```

**Example 8.4** Solving a hyperbolic partial differential equation

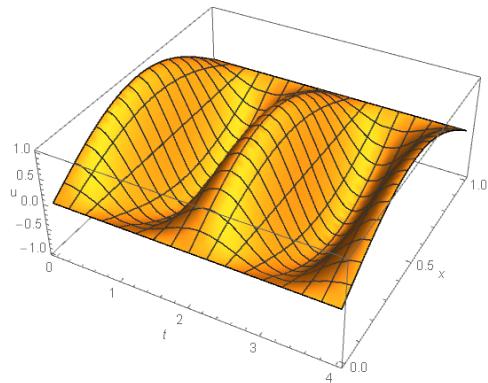
$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \quad 0 \leq x \leq 1, \quad 0 \leq t \leq 4$$

with the boundary conditions of $u(0,t) = u(1,t) = 0$ and the initial conditions of $u(x,0) = \sin(\pi x)$ and $\partial u / \partial t (x,0) = 0$.

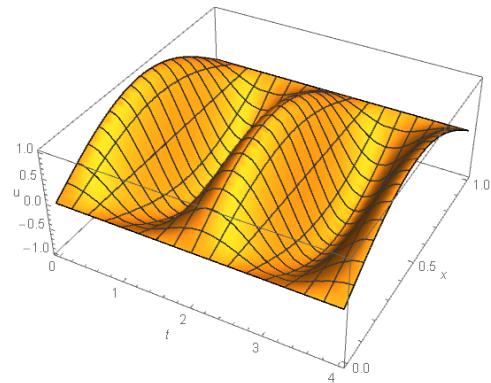
```

pde =  $\partial_{t,t}u[x, t] == \partial_{x,x}u[x, t]$ ;
ic = {u[0, t] == 0, u[1, t] == 0, u[x, 0] == Sin[\pi x],
      Derivative[0, 1][u][x, 0] == 0};
sol = NDSolve[{pde, ic}, u[x, t], {x, 0, 1}, {t, 0, 4}];
plotapprox = Plot3D[u[x, t] /. sol, {t, 0, 4}, {x, 0, 1},
                     AxesLabel -> {t, x, "u"}, Mesh -> 15];
(* Exact solution *)
uexact[x_, t_] = Sin[\pi x] Cos[\pi t];
plotexact = Plot3D[uexact[x, t], {t, 0, 4}, {x, 0, 1},
                    PlotRange -> All, AxesLabel -> {t, x, "u"}, Mesh -> 15];
GraphicsGrid[{{plotapprox, plotexact}}, ImageSize -> Large]

```



Numerical



Exact

Index

- Acceleration, 4, 20, 298
Adams-Bashforth method, 251
 formulas, 256
 fourth-order, 256
Adams-Moulton method, 258
 closed formulas, 258
 fourth-order, 259
Aerodynamic heating rate, 160
Air density, 111, 125,
Airfoil, 111, 125,
Approximate solution, 7
Approximation
 first-order, 32
 second-order, 33
 third-order, 33
 zero-order, 33
Atmospheric pressure, 166

Back substitution, 57, 59, 60, 63, 64
Backward divided differences, 211
Beam bending, 48
Bessel function, 112
Binary system, 12
Bisection method, 20, 22
Bits, 12
Boole's rule, 191, 206
Boundary condition
 Dirichlet, 272
 Neumann, 272
Boundary layer thickness, 161
Bracketing method, 24, 28, 39
Buckling, 16, 20, 47

C language, 9
Cable deflection, 50
CAE, 9
Central difference approximation, 285, 289
Central divided differences, 212
Cholesky decomposition method, 55, 78
Commercial software, 3, 22, 46, 208, 243
Complex geometry, 1, 305
Computer
 hardware, 8
 language, 9
 mainframe, 9
 personal, 9
 software, 1, 9, 13
Conjugate gradient method, 55, 88
 asymmetric matrix, 78, 97,
 bi-conjugate, 100
 generalized minimal residual, 100
 modified, 94
 quasi-minimal residual, 100
 square, 100
Conservation of energy, 54, 269, 273, 283
Continuous function, 112
Convergence, 23, 26,
 criterion, 23, 26
Coordinate transformation, 202
Corrector, 235, 254
Courant number, 300
Cramer's rule, 55, 101
Crank-Nicolson method, 292, 305

Dependent variable, 5, 39, 156, 210, 227

- Derivative
 - Approximate, 210
 - exact, 37, 211
- Determinant, 55, 63
- Differences
 - backward, 215
 - central, 215
 - forward, 215
- Differentiation, 174, 210
- Direct iteration method, 41
- Divergence, 32, 35
- Divided differences
 - central, 212,
 - first backward, 211, 255
 - first forward, 211
 - second forward, 213
- Drag coefficient, 4, 228
- Elliptic integral
 - first kind, 174, 218
- Error
 - absolute, 36
 - approximate, 12
 - blunder, 11
 - data, 11
 - exact, 179, 232
 - modeling, 11
 - percentage, 12
 - propagation, 11
 - relative, 36, 195
 - round-off, 11
 - true, 12
 - truncation, 11
- Euler's method, 229, 230, 260
- Exact
 - solution, 3, 5
 - temperature, 54, 277, 294
- Experimental data, 111, 142
- Explicit method, 284, 297
- Exponential
 - function, 14
 - model, 148
- Extrapolation, 111, 134
- False-position method, 20, 25
- Finite difference
 - mesh, 2
 - method, 1, 270, 299
 - model, 54, 277
- Finite element
 - mesh, 2
 - method, 1
- Finite volume method, 1
- Floating points, 12
- Flow rate, 140, 168
- Fortran language, 321
- Forward
 - differencing, 178
 - divided differences, 112
 - elimination, 58
- Fourier's law, 273, 283
- Function
 - cosine, 10, 329
 - error, 16
 - exponential, 14, 166,
 - hyperbolic cosine, 19
 - sine, 10, 277
- Gauss elimination method
 - improved, 63
 - naive, 61
 - problems, 62
- Gauss integration, 197
 - n -point, 200
 - two-point, 200
- Gauss quadrature, 174, 197
 - locations, 204
 - weights, 204, 208
- Gauss-Jordan method, 55, 68
- Gauss-Legendre formulas, 201
- Gauss-Seidel iteration method, 55, 85, 277
- Gradient, 216
- Graphical method, 20
- Gravitational acceleration constant, 20, 227
- Grid point, 55, 274, 280
- Heat conduction
 - bar, 283
 - plate, 305, 311
- Heat flux, 273
- Heun's method, 234
 - non-self-starting, 253
- High-speed wind tunnel, 160, 171
- Identity matrix, 63, 320
- Ill-conditioned system, 63
- Implicit method, 289
- Increment function, 240
- Independent variable, 228, 270
- Infinite series, 10, 228

- Initial condition, 273, 300
- Insulated edge, 281
- Integrand, 174
- Integration
 - area under curve, 174
 - double, 205
 - error, 178 , 182, 196
 - error function, 174, 217
 - logarithmic function, 174
 - polynomial, 188, 217
 - summation, 174
- Interpolation, 112
 - fourth-order, 124
 - linear, 119,112
 - quadratic, 113, 118
- Interval, 279
- Jacobi iteration method, 82
- Jacobian matrix, 44
- Lagrange interpolating
 - polynomials, 118
 - functions, 119
- Laplace's equation, 270, 274
- Least-squares regression, 141
- Linear regression, linear data, 142, 146
- Logarithm, 149, 162
 - natural, 149
- LU decomposition method, 55, 72, 277
- Mach number, 305
- Mass density, 283
- Mathematica, 9
- MATLAB, 9
 - built-in functions, 38, 332, 336
 - contour plotting, 339
 - decision commands, 349
 - environment, 325
 - function file, 344
 - graph plotting, 336
 - icon symbol, 331
 - input/output commands, 344
 - iteration commands, 352
 - long commands, 334
 - long format, 331
 - mathematica operations, 335
 - m-file, 251, 327
 - programming commands, 349
 - programming, 347
 - read/write data files, 344
- scalar variables, 328
- script file, 328
- short format, 333
- vector variables, 331
- MATLAB command
 - cumtrapz, 209
 - dblquad, 209
 - gradient, 216
 - ode113, 250
 - ode23, 250
 - ode45, 250
 - solver, 250
 - sysode, 252
 - trapz, 208
 - triplequad, 209
 - \(back slash), 72
- MATLAB function
 - chol, 81
 - fzero, 39
 - inline, 39
 - interp1, 133
 - lu, 83
 - polyfit, 154
 - polyval, 155
 - roots, 40
 - spline, 132
- MATLAB window
 - command, 324
 - edit/debug, 326
 - figure, 330
 - help, 330
 - history, 329
 - workspace, 330
- Matrix, 319
 - addition, 321
 - calculus, 323
 - coefficients, 320
 - column, 321
 - differentiation, 324
 - identity, 320
 - integration, 324
 - inverse, 322
 - inversion method, 71
 - multiplication, 326
 - non-positive definite, 100
 - partitioning, 323
 - positive definite, 91
 - row, 320
 - square, 320

- subtraction, 321
- symmetric, 320
- transpose, 322
- vector, 331
- Minimization, 143, 151
- Modified Euler's method, 237
- Multiple integration, 205
 - area, 205
 - volume, 205
- Multiple regression
 - linear data, 156
 - nonlinear data, 162
- Multistep methods, 252
- Newton's divided differences, 112
- Newton's second law, 4, 227, 298
- Newton-Cotes formulas, 188, 192, 198
- Newton-Raphson method, 32, 43
- Numerical integration, 173
- Numerical methods
 - advantages, 4
 - basic concept, 3
- Oblique shock, 47
- One-point iteration method, 20, 28
- One-step method, 252
- Open method, 28
- Ordinary differential equation, 227
 - coefficients, 228
 - coupled, 246
 - first-order, 227, 239
 - higher-order, 228
 - linear, 4, 228, 260
 - nonlinear, 228, 260
 - second-order, 228, 264, 367
 - system, 252
- Orthogonal property, 96
- Partial differential equation, 227, 269
 - coefficients, 272
 - elliptic, 270, 271
 - hyperbolic, 271, 297
 - linear, 270
 - nonlinear, 270
 - parabolic, 271, 283
 - second-order, 270, 285
- Pascal language, 9
- Pi, 12
- Pivoting, 65
- Polynomial
 - first-order, 112
 - n^{th} -order, 115, 135
 - second-order, 19, 120
- Positive definite, 88
- Power equation, 146
- Predictor, 235, 253
- Pressure head, 168
- Pressure, 156, 168
- Quadratic function, 89
- Regression
 - linear, 142
 - multiple, 142, 156
 - polynomial, 146, 150
- Remainder, 178
- Residual vector, 44, 92
- Reynolds number, 161, 166
- Romberg integration, 174, 178, 192
- Root of equation, 19
- Roots of nonlinear equations, 41, 51
- Runge's function, 123, 132
- Runge-Kutta formula
 - derivation, 355
- Runge-Kutta method, 255
 - first-order, 240
 - fourth-order, 243, 247, 355
 - second-order, 240
 - third-order, 242
- Saturation-growth-rate equation, 149
- Scaling, 63
- Search direction vector, 92
- Secant method, 20, 37
- Second-order derivative, 213, 227
- Separation of variables, 5, 231
- Shock wave
 - bow, 111
 - propagation, 297
- Significant figures, 12
- Simpson's rule, 176, 184
 - approximate error, 180, 182
 - composite, 188
 - error, 183, 218
 - segment, 174
- Simpson's three-eighth rule, 189, 191
 - error, 189
- Simultaneous equations, 53, 151, 276
- Slope, 244, 248, 252
 - average, 234, 237

- Solution
 - accuracy, 239, 240
 - approximate, 174, 296
 - diverged, 289, 294
 - exact, 228, 231
 - nonlinear, 250
- Space shuttle, 4
 - tiles, 161
- Spacing, 289
- Specific heat, 47, 152, 155, 285
- Specified
 - heating, 270
 - tolerance, 83, 279
- Spline interpolation, 112, 124, 131
 - cubic, 128
 - linear, 125
 - quadratic, 126
- Stencil form, 275, 281
- Step size, 216, 230, 333
- Stopping tolerance, 23
- Stress-strain data, 167, 169
- Successive over-relaxation method, 55, 87
- Supercomputer, 9
- Supersonic flow, 125
- Surface heating, 53
- Swinging pendulum, 174, 227, 246, 249
- System of equations, 44, 55
- Taylor series, 32, 43, 240, 255, 355
 - two variables, 231, 341
- Temperature, 269
 - transient, 16
- Thermal conductivity, 54, 273, 283
 - coefficient, 273, 283
- Time, 4
- Time step, 7, 243, 260, 285, 293
 - critical, 286, 291
- Total error
 - minimization, 142, 151, 163
- Transcendental equation, 20,
- Transient heat conduction, 271
- Trapezoidal rule, 174, 175
 - approximate error, 180, 182
 - composite, 180
 - error, 176, 177, 180
 - segment, 180
- Tridiagonal system, 67, 290
- True error, 186, 188
- Velocity, 4
- Vibration of string, 288, 313
- Weighting factor, 87
- Weighting functions, 33

