SOLVING APPLIED MATHEMATICAL PROBLEMS WITH MATLAB®

DINGYÜ XUE
YANGQUAN CHEN

CRC Press
Taylor & Francis Group
A CHAPMAN & HALL BOOK

MATLAB® examples

SOLVING APPLIED MATHEMATICAL PROBLEMS WITH MATLAB®

This page intentionally left blank

SOLVING APPLIED MATHEMATICAL PROBLEMS WITH MATLAB®

DINGYÜ XUE

YANGQUAN CHEN

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

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

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

International Standard Book Number-13: 978-1-4200-8250-0 (Hardcover)

Library of Congress Cataloging-in-Publication Data

Xue, Dingyu.
Solving applied mathematical problems with MATLAB / Dingyu Xue, YangQuan Chen.
p. cm.
Includes bibliographical references and index.
ISBN-13: 978-1-4200-8250-0
ISBN-10: 1-4200-8250-7
1. Engineering mathematics--Data processing. 2. MATLAB. 3. Numerical analysis--Data processing. 4. Mathematical optimization--Data processing. I. Chen, YangQuan, 1966- II. Title.

TA331.X84 2009
510.285'5133--dc22 2008025953

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

Preface

Computational Thinking,[1] coined and promoted by Jeannette Wing of Carnegie Mellon University, is getting more and more attention. "It represents a universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" as acknowledged by Dr. Wing, "Computational Thinking draws on math as its foundations." The present book responds to "Computational Thinking" by offering the readers enhanced math problem solving ability and therefore, the readers can focus more on "Computational Thinking" instead of "Computational Doing."

The breadth and depth of one's mathematical knowledge might not match his or her ability to solve mathematical problems. In today's applied science and applied engineering, one usually needs to get the mathematical problems at hand solved efficiently in a timely manner without complete understanding of the numerical techniques involved in the solution process. Therefore, today, arguably, it is a trend to focus more on how to formulate the problem in a form suitable for computer solution and on the interpretation of the results generated from the computer. We further argue that, even without a complete preparation of mathematics, it is possible to solve some advanced mathematical problems using a computer. We hope this book is useful for those who frequently feel that their level of math preparation is not high enough because they still can get their math problems at hand solved with the encouragement gained from reading this book.

Using computers to solve mathematical problems today is ubiquitous. MATLAB®/Simulink is considered as the dominant software platform for applied math related topics. Sometimes, one simply does not know one's problem could be solved in a much simpler way in MATLAB or Simulink. From what Confucius wrote, "The craftsman who wishes to work well has first to sharpen his implements,"[2] it is clear that MATLAB is the right, already sharpened "implement." However, a bothering practical problem is this: MATLAB documentation only shows "this function performs this," and what a user with a mathematical problem at hand wants is, "Given this math problem, through what reformulation and then use of what functions will get the problem solved." Frequently, it is very easy for one to get lost in thousands of functions offered in MATLAB plus the same amount, if not more, of functions contributed by the MATLAB users community. Therefore,

[1] `http://www.cs.cmu.edu/afs/cs/usr/wing/www/Computational_Thinking.pdf`

[2] Confucius. `http://www.confucius.org/lunyu/ed1509.htm`.

the major contribution of this book is to bridge the gap between "problems" and "solutions" through well grouped topics and tightly yet smoothly glued MATLAB example scripts and reproducible MATLAB-generated plots.

A distinguishing feature of the book is the organization and presentation of the material. Based on our teaching, research and industrial experience, we have chosen to present the course materials following the sequence

- Computer Mathematics Languages — An Overview
- Fundamentals of MATLAB Programming
- Calculus Problems
- Linear Algebra Problems
- Integral Transforms and Complex Variable Functions
- Nonlinear Equations and Optimization Problems
- Differential Equations Problems
- Data Interpolation and Functional Approximation Problems
- Probability and Statistics Problems
- Nontraditional Methods

In particular, in the nontraditional mathematical problem solution methods, we choose to cover some interesting and practically important topics such as set theory and fuzzy inference system, neural networks, wavelet transform, evolutional optimization methods including genetic algorithms and particle swarm optimization methods, rough set based data analysis problems, fractional-order calculus (derivative or integral of non-integer order) problems, etc., all with extensive problem solution examples. A dedicated CAI (computer aided instruction) kit including more than 1,300 interactive PowerPoint slides has been developed for this book for both instruction and self-learning purposes.

We hope that readers will enjoy playing with the scripts and changing them as they wish for a better understanding and deeper exploration with reduced efforts. Additionally, each chapter comes with a set of problems to strengthen the understanding of the chapter contents. It appears that the book is presenting in certain depth some mathematical problems. However, the ultimate objective of this book is to help the readers, after understanding *roughly* the mathematical background, to avoid the tedious and complex technical details of mathematics and find the reliable and accurate solutions to the interested mathematical problems with the use of MATLAB computer mathematics language. There is no doubt that the readers' ability to tackle mathematical problems can be significantly enhanced after reading this book.

This book can be used as a reference text for almost all college students, both undergraduates and graduates, in almost all disciplines which require certain levels of applied mathematics. The coverage of topics is practically broad yet with a balanced depth. The authors also believe that this book will be a good desktop reference for many who have graduated from college and are still involved in solving mathematical problems in their jobs.

Apart from the standard MATLAB, some of the commercial toolboxes may be needed. For instance, the Symbolic Math Toolbox is used throughout the book to provide alternative analytical solutions to certain problems. Optimization Toolbox, Partial Differential Equation Toolbox, Spline Toolbox, Statistics Toolbox, Fuzzy Logic Toolbox, Neural Network Toolbox, Wavelet Toolbox, and Genetic Algorithm and Direct Search Toolbox may be required in corresponding chapters or sections. A lot of MATLAB functions designed by the authors, plus some third-party free toolboxes, are also presented in the book. For more information on MATLAB and related products, please contact

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098, USA
Tel: 508-647-7000
Fax: 508-647-7101
E-mail: `info@mathworks.com`
Web: `http://www.mathworks.com`

The writing of this book started more than 5 years ago, when a Chinese version[3] was published in 2004. Many researchers, professors and students have provided useful feedback comments and inputs for the newly extended English version. In particular, we thank the following professors: Xinhe Xu, Fuli Wang of Northeastern University; Hengjun Zhu of Beijing Jiaotong University; Igor Podlubny of Technical University of Kosice, Slovakia; Shuzhi Sam Ge of National University of Singapore, Wen Chen of Hohai University, China. The writing of some parts of this book has been helped by Drs. Feng Pan, Daoxiang Gao, Chunna Zhao and Dali Chen, and some of the materials are motivated by the talks with colleagues at Northeastern University, especially Drs. Xuefeng Zhang and Haibin Shi. The computer aided instruction kit and solution manual were developed by our graduate students Wenbin Dong, Jun Peng, Yingying Liu, Dazhi E, Lingmin Zhang and Ying Luo.

Moreover, we are grateful to the Editors, LiMing Leong and Marsha Hecht, CRC Press, Taylor & Francis Group, for their creative suggestions and professional help. The "Book Program" from The MathWorks Inc., in particular, Hong Yang, MathWorks, Beijing, Courtney Esposito, Meg Vuliez and Dee Savageau, is acknowledged for the latest MATLAB software and technical problem supports.

The authors are grateful to the following free toolbox authors, to allow the inclusion of their contributions in the companion CD:

Dr. Brian K Birge, for particle swarm optimization toolbox (PSOt)
John D'Errico, for `fminsearchbnd` Toolbox
Mr. Koert Kuipers for his BNB Toolbox
Dr. Johan Löfberg, University of Linköping, Sweden for YALMIP

[3]Xue D and Chen Y Q, *Advanced applied mathematical problem solutions using MATLAB*, Beijing: Tsinghua University Press, 2004

Mr. Xuefeng Zhang, Northeastern University, China for RSDA Toolbox

Last but not least, Dingyü Xue would like to thank his wife Jun Yang and his daughter Yang Xue; YangQuan Chen would like to thank his wife Huifang Dou and his sons Duyun, David and Daniel, for their patience, understanding and complete support throughout this work.

Dingyü Xue
Northeastern University
Shenyang, China
`xuedingyu@mail.neu.edu.cn`

YangQuan Chen
Utah State University
Logan, Utah, USA
`yqchen@ieee.org`

Chapter 1

Computer Mathematics Languages — An Overview

1.1 Computer Solutions to Mathematics Problems

1.1.1 Why should we study computer mathematics language?

We all know that manual derivation of solutions to mathematical problems is a useful skill when the problems are not so complicated. However, for a great variety of mathematical problems, manual solutions are laborious or even not possible. Thus computers must be employed for solving these problems. There are basically two ways in solving these problems by computers. One is to verbally implement the existing numerical algorithms using general purpose computer languages such as Fortran or C. The other way is to use specific computer languages with a good reputation. These languages include MATLAB, Mathematica and Maple. In this book, they are referred to as the *computer mathematics languages*. The numerical algorithms can only be used to handle computation problems by numbers, while for problems like to find the solutions to the symbolic equation $x^3 + ax + c = d$, where a, c, d are not given numerical values but symbolic variables, the numerical algorithms cannot be used. The computer mathematics languages with symbolic computation capabilities should be used instead.

Before systematically introducing the contents of the book, the following examples are given such that the readers may understand and appreciate the necessity of using the computer mathematics languages.

Example 1.1 In calculus courses, the concepts and derivation methods are introduced with an emphasis on manual deduction and computation. If a function $f(x)$ is given by $f(x) = \dfrac{\sin x}{x^2+4x+3}$, how could one derive $\dfrac{\mathrm{d}^4 f(x)}{\mathrm{d}x^4}$ manually? Of course, one can derive it using the methods taught in calculus courses. For instance, the first-order derivative $\mathrm{d}f(x)/\mathrm{d}x$ can be derived first, the second-order derivative, third-order derivative and finally fourth-order derivative of the function $f(x)$ can be evaluated in turn. In this way, even higher-order derivatives of the function can be derived manually, in theory. However, the procedure is more suitable to be carried out with computers. With suitable computer mathematics languages, the fourth-order derivative of the function $f(x)$ can be calculated using a single statement as follows:

$$\frac{\sin x}{x^2+4x+3}+4\frac{(2x+4)\cos x}{(x^2+4x+3)^2}-12\frac{(2x+4)^2\sin x}{(x^2+4x+3)^3}+12\frac{\sin x}{(x^2+4x+3)^2}-24\frac{(2x+4)^3\cos x}{(x^2+4x+3)^4}$$
$$+48\frac{(2x+4)\cos x}{(x^2+4x+3)^3}+24\frac{(2x+4)^4\sin x}{(x^2+4x+3)^5}-72\frac{(2x+4)^2\sin x}{(x^2+4x+3)^4}+24\frac{\sin x}{(x^2+4x+3)^3}.$$

Of course, with built-in symbolic expression simplification methods, an even simpler form can be automatically derived for $\dfrac{\mathrm{d}^4 f(x)}{\mathrm{d}x^4}$ as follows:

$$(x^8+16x^7+72x^6-32x^5-1094x^4-3120x^3-3120x^2+192x+1581)\frac{\sin x}{(x^2+4x+3)^5}$$
$$+8(x^5+10x^4+26x^3-4x^2-99x-102)\frac{\cos x}{(x^2+4x+3)^4}.$$

It is obvious that manual derivation could be a tedious and laborious work and it could be quite complicated. Wrong results may be obtained even with a slightly careless manipulation of formulae. Thus, even though the results can be obtained, the results may be suspicious and untrustworthy. If the computer mathematics languages are used, the tedious and unreliable work can be avoided. For example, by using MATLAB language, the accurate $\mathrm{d}^{100}f(x)/\mathrm{d}x^{100}$ can be obtained in a second!

Example 1.2 In many fields, the roots of polynomial equations are often needed. The well-known Abel-Ruffini Theorem states that there is no general solution in radicals to polynomial equations of degree five or higher. The problems can be solved numerically using the Lin-Bairstrow algorithm. Now consider a polynomial equation

$$s^6+9s^5+\frac{135}{4}s^4+\frac{135}{2}s^3+\frac{1215}{16}s^2+\frac{729}{16}s+\frac{729}{64}=0.$$

Applying the Lin-Bairstrow method, under double-precision, the roots can be found as

$$s_{1,2}=-1.5056\pm \mathrm{j}0.0032,\quad s_{3,4}=-1.5000\pm \mathrm{j}0.0065,\quad s_{5,6}=-1.4944\pm \mathrm{j}0.0032.$$

Substituting s_1 back to the original equation, the error can be found to be $-8.7041\times 10^{-14}-\mathrm{j}1.8353\times 10^{-15}$. In fact, all the roots to the above equation are exactly -1.5, if the symbolic facilities of the computer mathematics languages are used.

Example 1.3 In linear algebra courses, the determinant of a matrix is suggested to be evaluated by algebraic complements. For instance, for an $n\times n$ matrix, its determinant can be evaluated from determinants of n matrices of size $(n-1)\times(n-1)$. Similarly, the determinant of each $(n-1)\times(n-1)$ matrix can be obtained from determinants of $n-1$ matrices of size $(n-2)\times(n-2)$. In other words, the determinant of an $n\times n$ matrix can finally be obtained from determinants of 1×1 matrices, i.e., the scalar itself. Thus, it can be concluded that the analytical solutions to the determinant of any given matrix exists.

In fact, the above mathematical conclusion neglected the computability and feasibility issue. The computation load for such an evaluation task could be extremely tremendous, which requires $(n-1)(n+1)!+n$ operations. For instance, when $n=20$, the number of floating-point operations (flops) for the computation is 9.7073×10^{20}, which amounts to 300 years of computation on mainframe computers of a billion flops per second. Thus, the algebraic complement method, although elegant and instructive, is not practically feasible. In real applications, the determinants of even larger sized matrices are usually needed ($n \gg 20$), which is clearly not possible to directly apply the algebraic complement method mentioned above.

In numerical analysis courses, various algorithms have been devised. However, due to finite precision numerical computation, these algorithms may have numerical problems when the matrix is close to being singular. For example, consider the Hilbert matrix given by

$$\boldsymbol{H} = \begin{bmatrix} 1 & 1/2 & 1/3 & \cdots & 1/n \\ 1/2 & 1/3 & 1/4 & \cdots & 1/(n+1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1/n & 1/(n+1) & 1/(n+2) & \cdots & 1/(2n-1) \end{bmatrix}. \tag{1.1}$$

For $n = 20$, an erroneous determinant $\det(\boldsymbol{H}) = 0$ could actually be obtained even if double-precision is used. On the other hand, if computer mathematics language MATLAB is used, the analytical solution below can be obtained within 0.4 seconds:

$$\det(\boldsymbol{H}) = \frac{1}{\underbrace{2377454\cdots 68000000000000000000000000000000000}_{\text{225 digits, with some digits omitted}}} \approx 4.2062 \times 10^{-224}.$$

Example 1.4 Consider the well-known nonlinear Van der Pol equation

$$\ddot{y} + \mu(y^2 - 1)\dot{y} + y = 0$$

and when μ is large, i.e., $\mu = 1000$, the conventional numerical algorithms for solving differential equations such as the standard Runge-Kutta method may cause numerical problems. Specialized numerical algorithms for stiff ordinary differential equations (ODEs) should be used instead, rather than the standard Runge-Kutta methods in numerical analysis courses.

As another example, the first-order delay differential equation (DDE)

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = -0.1y(t) + 0.2\frac{y(t-30)}{1 + y^{10}(t-30)}$$

cannot be solved using the commonly taught algorithms in numerical analysis courses. The MATLAB function `dde23()` or block diagram modeling tool Simulink can be used instead. The details of the methods will be given later in the book (Section 7.3).

Example 1.5 Consider the linear programming problem given below

$$\min_{\boldsymbol{x} \text{ s.t. } \begin{cases} 2x_2+x_3+4x_4+2x_5 \leqslant 54 \\ 3x_1+4x_2+5x_3-x_4-x_5 \leqslant 62 \\ x_1, x_2 \geqslant 0, x_3 \geqslant 3.32, x_4 \geqslant 0.678, x_5 \geqslant 2.57 \end{cases}} (-2x_1 - x_2 - 4x_3 - 3x_4 - x_5).$$

Since the original problem is a linear constrained optimization problem, the analytical unconstrained method, i.e., setting the derivatives of the objective function with respect to each decision variable x_i to zeros, cannot be used. With linear programming tools in MATLAB, the numerical solutions can be found easily as $x_1 = 19.7850,\ x_2 = 0,\ x_3 = 3.3200,\ x_4 = 11.3850,\ x_5 = 2.5700$.

Applying algorithms in numerical analysis or optimization courses, conventional constrained optimization problems can be solved. However, if other special constraints are introduced, for instance, the decision variables are constrained to be integers, the integer programming must be used. There are not so many books introducing softwares that can tackle the integer and mixed-integer programming problems. If we use MATLAB, the solutions to this example problem are easily found as $x_1 = 19, x_2 = 0, x_3 = 4, x_4 = 10, x_5 = 5$.

Example 1.6 In many other courses of applied mathematics branches, such as integral transform, complex variable functions, partial differential equations, data interpolation and fitting, probability and statistics, can you still remember how to solve the problems after the final exams?

In many subjects, such as electric circuits, electronics, power electronics, motor drive, automatic control theory, more sophisticated examples and problems are usually skipped due to the lack of introduction of high-level computer software tools. If computer mathematics languages are introduced routinely in the above courses, complicated practical problems can be solved and innovative solutions to the problems can be explored.

1.1.2 Analytical solutions versus numerical solutions

The development of modern sciences and engineering depends heavily on mathematics. However, the research interests of pure mathematicians are different from other scientists and engineers. Mathematicians are often more interested in finding the analytical or closed-form solutions to mathematical problems. They are in particular interested in proving the existence and uniqueness of the solutions, and do not usually care much about what the solutions are. Engineers and scientists are more interested in finding the exact or approximate solutions to the problems at hand and usually do not care too much about the details on how the results are obtained, as long as the results are reliable and meaningful. The most widely used approaches for finding the approximate solutions are the numerical techniques.

It is quite common to find that analytical solutions do not exist in reality in many different mathematics branches. For instance, it is well-known that the definite integral $\dfrac{2}{\sqrt{\pi}}\displaystyle\int_0^a \mathrm{e}^{-x^2}\mathrm{d}x$ has no analytical solution. To solve the problem, mathematicians introduce a special function $\mathrm{erf}(a)$ to denote it and do not care what in particular the numerical value is. In order to find an approximate value, scientists and engineers have to use numerical approaches.

Another example is that the irrational number π has no closed-form solution. The ancient Chinese astronomer and scientist Zu Chongzhi, also known as *Tsu Ch'ung-chih*, found that the value is between 3.1415926 and 3.1415927, in about A.D. 480. This value is accurate enough in most science and engineering practice. Even with the imprecise value 3.14 found by Archimedes in about B.C 250 (?), the solutions to most engineering problems are often acceptable.

The above discussions hint that an approximate numerical solution is ubiquitous. In many cases, only showing existence and uniqueness of solutions is not enough. We need to compute the solution using computers. The breadth and depth of one's mathematical knowledge might not match one's ability of getting mathematical problems solved. In today's applied science and applied engineering, one usually needs to get the mathematical problems at hand solved efficiently in a timely manner without complete understanding of the numerical techniques involved even in the solution process. Therefore, today, arguably, it is a trend to focus more on how to formulate the problem in a form suitable for computer solution and on the interpretation of the results generated from the computer. Numerical techniques have already been used in many scientific and engineering areas. For instance, in mechanics, finite element methods (FEM) have been used in solving partial differential equations. In aerospace and control, numerical linear algebra and numerical solutions to ordinary differential equations have successfully been used for decades. For simulation experiments in engineering and non-engineering areas, numerical solutions to difference and differential equations are the core problems. In hi-tech developments, digital signal processing based on fast Fourier transform (FFT) has been regarded as a routine task. There is no doubt that if one masters one or more practical computation tools, significant enhancement of mathematical problem solving capability can be expected.

1.1.3 Mathematics software packages: an overview

The emerging digital computers fueled the developments of numerical as well as symbolic computation techniques. In the early stages of the development of numerical computation techniques, some well established packages, such as the eigenvalue-based package EISPACK[1, 2], linear algebra package LINPACK[3] in the USA, the NAG package by the Numerical Algorithm Group in the UK, and the package in the well accepted book *Numerical Recipes*[4], appeared and were widely used with good user feedback.

The famous EISPACK and LINPACK packages are both specific packages for numerical linear algebra applications. Originally developed in the USA, EISPACK and LINPACK packages were written in Fortran. To have a flavor of how to use the packages, let us consider eigenvalues ($\boldsymbol{W}_{\rm R}$, $\boldsymbol{W}_{\rm I}$ for their real and imaginary parts) and eigenvectors $\boldsymbol{Z}$ of an $N \times N$ real matrix $\boldsymbol{A}$. As suggested by EISPACK, the standard solution method is by sequentially calling relevant subroutines provided in EISPACK as follows:

```
      CALL BALANC(NM,N,A,IS1,IS2,FV1)
      CALL ELMHES(NM,N,IS1,IS2,A,IV1)
      CALL ELTRAN(NM,N,IS1,IS2,A,IV1,Z)
      CALL HQR2(NM,N,IS1,IS2,A,WR,WI,Z,IERR)
      IF (IERR.EQ.0) GOTO 99999
```

```
      CALL BALBAK(NM,N,IS1,IS2,FV1,N,Z)
```

Apart from the main body of the program, the user should also write a few lines to input or initialize the matrix $\boldsymbol{A}$ to the above program and return or display the results obtained by adding some display or printing statements. Then, the whole program should be compiled and linked with the EISPACK library to generate an executable program. It can be seen that the procedure is quite complicated. Moreover, if another matrix is to be solved, the whole procedure might be repeated, which makes the solution process even more complicated.

It is good news that the mathematical software packages are continuously developing, implementing the leading-edge numerical algorithms, providing more efficient, more reliable, faster and more stable packages. For instance, in the area of numerical algebra, a new LaPACK is becoming the leading package. Unlike the original purposes of EISPACK or LINPACK, the objectives of LaPACK have been changed. LaPACK is no longer aiming at providing libraries or facilities for direct user applications. Instead, LaPACK provides support to mathematical software and languages. For example, MATLAB and a freeware Scilab have abandoned the packages of LINPACK and EISPACK, and adopted LaPACK as their low-level library support.

1.2 Summary of Computer Mathematics Languages

1.2.1 A brief historic review of MATLAB

In the late 1970's, Professor Cleve Moler, the Chairman of the Department of Computer Science at the University of New Mexico found that the solutions to linear algebraic problems using the most advanced EISPACK and LINPACK packages are too complicated. MATLAB (MATrix LABoratory) was then conceived and developed. The first release of MATLAB was freely distributed in 1980. Cleve Moler and Jack Little co-founded The MathWorks Inc. in 1984 to develop the MATLAB language. At that time, state-space-based control theory was rapidly developing and a significant amount of numerical algebra problems needed to be solved. The appearance of MATLAB and its Control Systems Toolbox soon attracted the attention of the control community. More and more control oriented toolboxes were written by distinguished experts in different control disciplines, which added higher reputations to MATLAB. It is true that MATLAB was initiated by numerical mathematicians but its impacts and innovations were first built by the control community. Soon it became the general purpose language of control scientists and engineers. With more and more new toolboxes in many other engineering disciplines, MATLAB is becoming the *de facto* standard language of science and engineering.

1.2.2 Three widely used computer mathematics languages

There are three leading computer mathematics languages in the world with high reputations. They are MATLAB of The MathWorks, Mathematica of Wolfram Research, and Maple of Waterloo Maple. They each have their own distinguishing merits, for instance, MATLAB is good at numerical computation and easy in programming, while Mathematica and Maple are powerful in pure mathematics problems involving symbolics and derivations.

The numerical computation capability of MATLAB is much stronger than the other two languages. Besides, various nice toolboxes by experts can be used to tackle the problems with high efficiency. In addition, the symbolic computation engine in Maple can be used to solve symbolic computation problems. Thus, the symbolic computation capability of MATLAB is essentially as good as Mathematica and Maple for most mathematical problems. When the readers have mastered such a computer mathematics language like MATLAB, the ability of handling mathematical problems could be enhanced significantly.

1.2.3 Introduction to free scientific open-source softwares

Although many extremely powerful scientific computation facilities have been provided in the computer mathematics languages such as MATLAB, Maple and Mathematica, there are certain limitations in their applications in research and education, for example, they are expensive commercial softwares. Moreover, some of the core source code are not accessible to the users. Thus the open-source softwares are welcome in scientific computation as well. Some influential softwares include:

(i) **Scilab** Scilab is developed and maintained by INRIA, France. The syntaxes are very similar with MATLAB. It is a free open-source software which concentrates in particular on control and signal processing. The Scicos in Scilab is a block diagram simulation environment similar to Simulink. The web-page of Scilab is `http://www.scilab.org/`.

(ii) **Octave** Octave was conceived in 1988 and first released in 1993. It is a promising open-source software for numerical computation, initiated from numerical linear algebra. The earlier objective of the software was to provide support in education. The web-page of Octave is `http://www.gnu.org/software/octave/`.

(iii) **Others** Some other small-scale numerical matrix computation softwares such as Freemat and SpeQ are all attractive free softwares. The web-pages are respectively `http://freemat.sourceforge.net/wiki/index.php/Main_Page` and `http://www.speqmath.com/index.php?id=1`.

1.3 Outline of the Book

The book can be used as a reference text or even a textbook of a new course on scientific computation. The applications of all branches of college mathematics can be taught in such a course with broad coverage, which enables the students view mathematics from a different angle. This will significantly increase the ability of the students for mathematical problem solutions. The book can also be used as a reference book for actual mathematical problem solutions.

The contents of the book are summarized below:

Chapter 1, the current chapter, gives an overview of the development of MATLAB and other computer mathematics languages.

Chapter 2, “Fundamentals of MATLAB Programming,” introduces briefly the programming essentials of MATLAB, including data structure, flow control structures and M-function programming. Two-dimensional and three-dimensional graphics are also presented. This chapter is the basis for the materials in the book.

Chapter 3, “Calculus Problems,” covers the problems in college calculus, from a different viewpoint. The subjects introduced in the chapter include limits, derivatives and integrals of single-variable and multivariable functions. Series expansion problems such as Taylor series and Fourier series expansions as well as series sums and products are covered. Numerical differentiation and integration or quadrature, are also introduced. Finally MATLAB solutions to path, line and surface integrals are illustrated.

Chapter 4, “Linear Algebra Problems,” studies linear algebra problems using both analytical and numerical methods. Special matrices in MATLAB are first discussed followed by basic matrix analysis, matrix transformation and matrix decomposition problems. Matrix equation solutions, including linear equations, Lyapunov equation and Riccati equations, are introduced. How to evaluate matrix functions is introduced for both the exponential function and the functions of arbitrary forms.

Chapter 5, “Integral Transforms and Complex Variable Functions,” includes the solutions to Laplace transform problems and their inverse, Fourier transforms and their variations, Z, Mellin and Hankel transforms. The analysis of complex variable functions are also introduced, including poles, residues, partial fraction expansion and closed-path integral problems, all with many illustrative solution examples.

Chapter 6, “Nonlinear Equations and Optimization Problems,” explores the search methods for linear equations, nonlinear equations and nonlinear matrix equations. The unconstrained optimization, constrained optimization and mixed integer programming problems are demonstrated. Linear matrix inequality (LMIs) problems are also covered in the chapter.

Chapter 7, “Differential Equations Problems,” mainly covers analytical as well as numerical solutions to ordinary differential equations. Different types of ordinary differential equations, including stiff equations, implicit equations, differential algebraic equations, delay differential equations and the boundary valued equations are illustrated. An introduction to partial differential equations is also given briefly through examples.

Chapter 8, "Data Interpolation and Functional Approximation Problems," studies the interpolation problems such as simple interpolation, cubic spline and B-spline problems. We show that numerical differentiation and integration problems can be solved with splines. Polynomial fitting, continued fraction expansion and Padé approximation as well as least squares curve fitting methods are all covered and illustrated. Fast Fourier transform, signal filtering and de-noising problems are also studied briefly in this chapter.

Chapter 9, "Probability and Mathematical Statistics Problems," studies the probability distributions and pseudo-random number generators first. Statistical analysis to the measured random data is then illustrated. Hypothesis tests for a few common applications are presented, and the analysis of variance method is also demonstrated briefly.

Chapter 10, "Nontraditional Solution Methods," covers a wide variety of interesting topics, such as traditional set theory, rough set theory, fuzzy set theory and fuzzy inference system, neural networks, wavelet transform, evolutional optimization methods including genetic algorithms and particle swarm optimization methods. Most interestingly, fractional-order calculus (derivative or integral of non-integer order) problems are introduced with basic numerical computational techniques and examples.

It appears that the book is presenting in certain depth some mathematical problems. However, the ultimate objective of this book is to help the readers, after understanding roughly the mathematical background, to avoid the tedious and complex technical details of mathematics and find the reliable and accurate solutions to the interested mathematical problems with the use of MATLAB computer mathematics language. There is no doubt that the readers' ability to tackle mathematical problems can be significantly enhanced after reading this book.

Exercises

1. Install MATLAB on your machine, and issue the command `demo`. From the dialog boxes and menu items of the demonstration program, experience the powerful functions provided in MATLAB.
2. Type the command `doc symbolic/diff`, and see whether it is possible, by reading the relevant help information, to solve the problem given in Example 1.1. If the solutions can be obtained, compare the solutions with the results in the example.
3. Solve the following Lyapunov equation by starting the command

```
lookfor lyapunov
```

and see whether there is any function related to the keyword `lyapunov`. If there is one, say, the `lyap` function is found, type `doc lyap` and see whether there is a way to solve this Lyapunov equation. Check the accuracy of the solution by back substitution.

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix} \boldsymbol{X} + \boldsymbol{X} \begin{bmatrix} 16 & 4 & 1 \\ 9 & 3 & 1 \\ 4 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}.$$

4. Write a simple subroutine which can be used to perform matrix multiplications in other languages such as the C language. Try to make the code as general purpose as possible.
5. Write a piece of code in C language which can generate the Fibonacci sequence defined as $a_1 = a_2 = 1$, $a_{k+2} = a_k + a_{k+1}$, for $k = 1, 2, \cdots$. Generate the sequence with 50 terms. Observe whether the results are feasible. If there are serious abnormal problems, are there any possible solutions in C?

Chapter 2

Fundamentals of MATLAB Programming

MATLAB language is becoming a widely accepted scientific language, especially in the field of automatic control. In other engineering and non-engineering disciplines such as economics and even biology, MATLAB is also an attractive and promising computer mathematics language. In this book, we shall concentrate on the introduction to MATLAB with its applications in solving applied mathematics problems. A good working knowledge of MATLAB language will enable one not only understand in depth the concepts and algorithms in research but also increase the ability to do creative research work and apply MATLAB to actively tackle the problems in other related courses.

As a programming language, MATLAB has the following advantages:

(i) **Clarity and high efficiency** MATLAB language is a highly integrated language. A few MATLAB sentences may do the work of hundreds of lines of source code of other languages. Thus the MATLAB program is more reliable and easy to maintain.

(ii) **Scientific computation** The basic element in MATLAB is a complex matrix of double-precision. Matrix manipulations can be carried out directly. Numerical computation functions provided in MATLAB, such as the ones for solving optimization problems or other mathematical problems, can be used directly. Also symbolic computation facilities are provided in MATLAB's Symbolic Math Toolbox to support formula derivation.

(iii) **Graphics facilities** MATLAB language can be used to visualize the experimental data in an easy manner. Moreover, the graphical user interface is also supported in MATLAB.

(iv) **Comprehensive toolboxes and blocksets** There are a huge amount of MATLAB toolboxes and Simulink blocksets contributed by experienced programmers and researchers.

(v) **Powerful simulation facilities** The powerful block diagram-based modeling technique provided in Simulink can be used to analyze systems with almost any complexity. In particular, under Simulink, the control blocks, electronic blocks and mechanical blocks can be modeled together under the same framework, which is currently not possible in other computer mathematics languages.

In Section 2.1, the fundamental information about MATLAB programming, such as the data types, statement structures, colon expressions and sub-matrix extraction is introduced. In Section 2.2, the basic operations, including algebraic, logic and relationship operations, and simplification of symbolic formulae, and introduction to number theory are presented. The flow control such as loop structures, conditional structures, switches and trial structures are introduced in Section 2.3. In Section 2.4, the most important programming structure — the M-function — is illustrated with useful hints on high-level programming. In Section 2.5, two-dimensional graphics facilities are presented, where two-dimensional sketching and implicit function expressions are illustrated in particular. Three-dimensional graphics are presented in Section 2.6, where mesh and surface plots can be drawn and the viewpoint setting facilities are introduced.

2.1 Fundamentals of MATLAB Programming

2.1.1 Variables and constants in MATLAB

MATLAB variable names consist of a letter, followed by any number of letters, digits, or underscores. For instance, `MYvar12`, `MY_Var12` and `MyVar12_` are valid variable names, while `12MyVar` and `_MyVar12` are invalid ones, since the first character is not a letter. The variable names are case-sensitive, i.e., the variables `Abc` and `ABc` are different variables.

In MATLAB, some of the names are reserved for the constants. They can however be assigned to other values. It is suggested that these names should not be assigned to other values whenever possible.

- `eps` — error tolerance for floating-point operation. The default value is `eps`$= 2.2204 \times 10^{-16}$, and if the absolute value of a quantity is smaller than `eps`, it can be regarded as 0.
- `i` and `j` — If `i` or `j` is not overwritten, they both represent $\sqrt{-1}$. If they are overwritten, they can be restored with the `i=sqrt(-1)` command.
- `Inf` — the MATLAB representation of infinity quantity $+\infty$. It can also be written as `inf`. Similarly $-\infty$ can be written as `-Inf`. When 0 is used in denominator, the value `Inf` can be generated, with a warning. This agrees with the IEEE standard. For mathematical computation, this definition has its advantages over C language.
- `NaN` — not a number, which is often returned by the operations 0/0, `Inf/Inf` and others. It should also be noted that `NaN` times `Inf` will return `NaN`.
- `pi` — double-precision representation of the circumference ratio π.

- `lasterr` — returns the error message received last time. It can be a string variable, with empty string for no error message generated.
- `lastwarn` — returns the last obtained warning message.

2.1.2 Data structure

Double-precision data type

Numerical computation is the most widely used computation form in MATLAB. To ensure high-precision computations, double-precision floating-point data type is used, which is 8 bytes (64 bits). According to the IEEE standard, it is composed of 11 exponential bits, 53 number bits and a sign bit, representing the data range of $\pm 1.7\times 10^{308}$. The MATLAB function for defining this data type is `double()`. In other special applications, i.e., in image processing, unsigned 8 bit integer can be used, whose function is `uint8()`, representing the value in $(0, 255)$. Thus significant memory space is saved. Also the data types such as `int8()`, `int16()`, `int32()`, `uint16()` and `uint32()` can be used.

Symbolic data type

"Symbolic" variables are also defined in MATLAB in contrast to the numerical variables. They can be used in formula derivation and analytical solutions of mathematical problems. Before finding analytical solutions, the related variables should be declared as `symbolic`, with the `syms` statement `syms` *var_list var_props*, where *var_list* is the list of variables to be declared, separated by spaces. If necessary, the types of the properties of the variables can be assigned by *var_props*, such as `real` or `positive`. For instance, if one wants to assume that `a,b` are symbolic variables, the statement `syms a b` can be used. Also the statement `syms a nonzero` can be used to say that `a` is a nonzero variable.

The variable precision arithmetic function `vpa()` can be used to display the symbolic variables in any precision. The syntax of the function is `vpa(`A`,`n`)` or `vpa(`A`)`, where A is the variable to be displayed, and n is the number of digits expected, with the default value of 32 decimal digits.

Example 2.1 Display the first 300 digits of π.

Solution The following statement can be used directly to display the exact value of π

```
>> vpa(pi,300)
```

and the result is shown as 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034

4861045432664821339360726024914127.

One may also require large number of digits to be displayed. Also the result obtained with the statement `vpa(pi)` is 3.14159265358979323846264338327 95.

Other data types

Apart from the commonly used numerical data types in MATLAB, the following data types are also provided such that

(i) **Strings** String variables are used to store messages. The syntax of string is slightly different from that in C; single quotation marks are used in MATLAB.

(ii) **Multi-dimensional arrays** Three-dimensional arrays are the direct extension of matrices. Multi-dimensional arrays are also provided in MATLAB.

(iii) **Cell arrays** Cells are extension of matrices, whose elements are no longer values. The element, referred to as *cells*, of cell arrays can be of any data type. For instance, $A\{i,j\}$ can be used to represent the (i,j)th term of cell array A.

(iv) **Classes and objects** MATLAB allows the use of classes in the programming. For instance, the transfer function class in control can be used to represent a transfer function of a system in one single variable. An example of the creation and overload function programming of an object is given in Section 10.6.

2.1.3 Basic structure of MATLAB

Two types of MATLAB statements can be used:

(i) **Direct assignment** The basic structure of this type of statement is `variable = expression`,
and *expression* can be evaluated and assigned to the variable defined in the left-hand-side, and established in MATLAB workspace. If there is a semicolon used at the end of the statement, the result is not displayed. Thus the semicolon can be used to suppress the display of intermediate results. If the left-hand-side variable is not given, the expression will be assigned to the reserved variable `ans`. Thus, the reserved variable `ans` always stores the latest statements without a left-hand-side variable.

Example 2.2 Specify the matrix $\boldsymbol{A}=\begin{bmatrix}1&2&3\\4&5&6\\7&8&0\end{bmatrix}$ into MATLAB workspace.

Solution The matrix $\boldsymbol{A}$ can easily be entered into MATLAB workspace, with the following statement

```
>> A=[1,2,3; 4 5,6; 7,8 0]
```

where >> is the MATLAB prompt, which is given automatically in MATLAB. Under the prompt, various MATLAB commands can be specified. For matrices, square brackets should be used to describe matrices, with the elements in the same row separated by commas, and the rows are separated by semicolons. The double matrix variable $\boldsymbol{A}$ can then be established in MATLAB workspace. The matrix $\boldsymbol{A}$ can be displayed in MATLAB command window

```
A =
     1     2     3
     4     5     6
     7     8     0
```

A semicolon at the end of the statement suppresses the display of such a matrix. The size of a matrix can be expanded or reduced dynamically, with the following statements.

```
>> A=[1,2,3; 4 5,6; 7,8 0];       % assignment is made, however no display
   A=[[A; [1 2 3]], [1;2;3;4]]; % dynamically define the size of matrix
```

Example 2.3 Enter complex matrix $\boldsymbol{B} = \begin{bmatrix} 1+\mathrm{j}9 & 2+\mathrm{j}8 & 3+\mathrm{j}7 \\ 4+\mathrm{j}6 & 5+\mathrm{j}5 & 6+\mathrm{j}4 \\ 7+\mathrm{j}3 & 8+\mathrm{j}2 & 0+\mathrm{j}1 \end{bmatrix}$ into MATLAB.

Solution Specifying a complex matrix in MATLAB is as simple as with the case for real matrices. The notations `i` and `j` can be used to describe the imaginary unit. Thus the following statement can be used to enter the complex matrix $\boldsymbol{B}$

```
>> B=[1+9i,2+8i,3+7j; 4+6j 5+5i,6+4i; 7+3i,8+2j 1i]
```

(ii) **Function call statement** The basic statement structure of function call is

[*returned_arguments*]=*function_name*(*input_arguments*)

where, the regulation for function names are the same as in variable names. Generally the function names are the file names in the MATLAB path. For instance, the function name `my_fun` corresponds to the file my_fun.m. Of course, some of the functions are built-in functions in MATLAB kernel, such as the `inv()` function.

More than one input arguments and returned arguments are allowed, in which case, commas should be used to separate the arguments. For instance, the function call [$\boldsymbol{U}$ $\boldsymbol{S}$ $\boldsymbol{V}$]=`svd`($\boldsymbol{X}$) performs singular value decomposition to a given matrix $\boldsymbol{X}$, and the three arguments $\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}$ will be returned.

2.1.4 Colon expressions and sub-matrices extraction

Colon expression is an effective way in defining row vectors. It is useful in generating vectors, and in extracting sub-matrices. The typical form of colon expression is $\boldsymbol{v}$=s_1:s_2:s_3. Thus a row vector $\boldsymbol{v}$ can be established in MATLAB workspace, with the initial value s_1, the increment s_2 and the final value s_3. If the term s_2 is omitted, a default increment of 1 is used instead. The examples given below illustrate the use of colon expressions.

Example 2.4 For different increments, establish vectors for $t \in [0, \pi]$.

Solution One may select an increment 0.2. The following statement can be used to establish a row vector such that

```
>> v1=0: 0.2: pi
```

and the row vector is then established such that $\boldsymbol{v}_1 = [0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2, 2.2, 2.4, 2.6, 2.8, 3]$. It is noted that the last term in $\boldsymbol{v}_1$ is 3, rather than π.

The following statements can be used to establish row vectors using colon expressions

```
>> v2=0: -0.1: pi  % negative step here means no vector generated
   v3=0:pi         % with the default step size of 1
   v4=pi:-1:0      % the new vector in the reversed order
```

thus $\boldsymbol{v}_2$ is a 1×0 (empty) matrix, $\boldsymbol{v}_3 = [0, 1, 2, 3]$, while $\boldsymbol{v}_4 = [3.142, 2.142, 1.142, 0.142]$.

The sub-matrix of a given matrix $\boldsymbol{A}$ can be extracted with the MATLAB statement, and the matrix can be extracted with `B=A(`$\boldsymbol{v}_1$`,`$\boldsymbol{v}_2$`)`, where $\boldsymbol{v}_1$ vector contains the numbers in the rows, and $\boldsymbol{v}_2$ contains the numbers of columns. Thus the relevant columns and rows can be extracted from matrix $\boldsymbol{A}$. The sub-matrix can be returned in matrix $\boldsymbol{B}$. If $\boldsymbol{v}_1$ is assigned to `:`, all the columns can be extracted. The keyword `end` can be used to indicate the last row or column.

Example 2.5 With the following statements, different sub-matrices can be extracted from the given matrix $\boldsymbol{A}$, such that

```
>> A=[1,2,3; 4,5,6; 7,8,0];
   B1=A(1:2:end, :)        % extract all the odd rows of matrix A
   B2=A([3,2,1],[1 1 1]) % copy the reversed first column to all columns
   B3=A(:,end:-1:1)        % flip left-right the given matrix A
```

and the sub-matrices extracted with the above statements are

$$\boldsymbol{B}_1 = \begin{bmatrix} 1 & 2 & 3 \\ 7 & 8 & 0 \end{bmatrix}, \ \boldsymbol{B}_2 = \begin{bmatrix} 7 & 7 & 7 \\ 4 & 4 & 4 \\ 1 & 1 & 1 \end{bmatrix}, \ \boldsymbol{B}_3 = \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 0 & 8 & 7 \end{bmatrix}.$$

2.2 Fundamental Mathematical Calculations

2.2.1 Algebraic operations of matrices

Suppose matrix $\boldsymbol{A}$ has n rows and m columns, it is then referred to as an $n \times m$ *matrix*. If $n = m$, then matrix $\boldsymbol{A}$ is also referred to as a *square matrix*. The following algebraic operations can be defined:

(i) **Matrix transpose** In mathematics textbooks, the transpose of matrices is often denoted as $\boldsymbol{A}^{\mathrm{T}}$. For an $n \times m$ matrix $\boldsymbol{A}$, the transpose matrix $\boldsymbol{B}$ can be defined as $b_{ji} = a_{ij}$, $i = 1, \cdots, n$, $j = 1, \cdots, m$, thus $\boldsymbol{B}$ is an $m \times n$ matrix. If matrix $\boldsymbol{A}$ contains complex elements, a special transpose can also be defined as $b_{ji} = a_{ij}^*$, $i = 1, \cdots, n$, $j = 1, \cdots, m$, i.e., the complex conjugate transpose matrix $\boldsymbol{B}$ is defined. This kind of transpose is referred to as the *Hermitian transpose*, denoted as $\boldsymbol{B} = \boldsymbol{A}^{\mathrm{H}}$. In MATLAB, `A'` can be used to evaluate the Hermitian matrix of $\boldsymbol{A}$. The simple transpose can be obtained with `A.'`. For a real matrix $\boldsymbol{A}$, `A'` is the same as `A.'`.

(ii) **Addition and subtraction** Assume that there are two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ in MATLAB workspace, the statements $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$ and $\boldsymbol{C} = \boldsymbol{A} - \boldsymbol{B}$ can be used respectively to evaluate the addition and subtraction of these two matrices. If the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are with the same size, the relevant results can be obtained. If one of the matrices is a scalar, it can be added to or subtracted from the other matrix. If the sizes of the two matrices are different, error messages can be displayed.

(iii) **Matrix multiplication** Assume that matrix $\boldsymbol{A}$ of size $n \times m$ and matrix $\boldsymbol{B}$ of size $m \times r$ are two variables in MATLAB workspace, and the columns of $\boldsymbol{A}$ equal the rows of $\boldsymbol{B}$, the two matrices are referred to as *compatible*. The product can be obtained from $c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}$, where $i = 1, 2, \cdots, n$, $j = 1, 2, \cdots, r$. If one of the matrices is a scalar, the product can also be obtained. In MATLAB, the multiplication of the two matrices can be obtained with `C=A*B`. If the two matrices are not compatible, an error message will be given.

(iv) **Matrix left division** The left division of the matrices `A\B` can be used to solve the linear equations $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$. If matrix $\boldsymbol{A}$ is non-singular, then $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$. If $\boldsymbol{A}$ is not a square matrix, `A\B` can also be used to find the least squares solution to the equations $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$.

(v) **Matrix right division** The statement `B/A` can be used to solve the linear equations $\boldsymbol{X}\boldsymbol{A} = \boldsymbol{B}$. More precisely, `B/A=(A'\B')'`.

(vi) **Matrix flip and rotation** The left-right flip and up-down flip of a given matrix $\boldsymbol{A}$ can be obtained with `B=fliplr(A)` and `C=flipud(A)` respectively, such that $b_{ij} = a_{i,n+1-j}$ and $c_{ij} = a_{m+1-i,j}$. The command `D=rot90(A)` rotates matrix $\boldsymbol{A}$ counterclockwise by 90°, such that $d_{ij} = a_{j,n+1-i}$.

(vii) **Matrix power** $\boldsymbol{A}^x$ computes the matrix $\boldsymbol{A}$ to the power x when matrix $\boldsymbol{A}$ is square. In MATLAB, the power can be evaluated with `F=A^x`.

(viii) **Dot operation** A class of special operation is defined in MATLAB. The statement `C=A.*B` can be used to obtain element-by-element product of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, such that $c_{ij} = a_{ij}b_{ij}$. The dot product is also referred to as the *Hadamard product.*

Dot operation plays an important role in scientific computation. For instance, if a vector $\boldsymbol{x}$ is given, then the vector $[x_i^5]$ cannot be obtained with `x^5`. Instead, the command `x.^5` should be used. In fact, some of the functions such as `sin()` can also be used in element-by-element operation.

Dot operation can be used to deal with other problems, for instance, the statement `A.^A` can be used, with the (i,j)th element then defined as $a_{ij}^{a_{ij}}$. Thus the matrix can be obtained

$$\begin{bmatrix} 1^1 & 2^2 & 3^3 \\ 4^4 & 5^5 & 6^6 \\ 7^7 & 8^8 & 0^0 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 27 \\ 256 & 3125 & 46656 \\ 823543 & 16777216 & 1 \end{bmatrix}.$$

Example 2.6 Consider again the matrix $\boldsymbol{A}$ in Example 2.2. Find all the cubic roots of such a matrix and verify the results.

Solution The cubic root of the matrix $\boldsymbol{A}$ can easily be found that

```
>> A=[1,2,3; 4,5,6; 7,8,0]; C=A^(1/3), e=norm(A-C^3)
```

and it can be found, with an error of $e = 8.2375\times10^{-15}$, that

$$\boldsymbol{C} = \begin{bmatrix} 0.77179 + \mathrm{j}0.6538 & 0.48688 - \mathrm{j}0.015916 & 0.17642 - \mathrm{j}0.2887 \\ 0.88854 - \mathrm{j}0.072574 & 1.4473 + \mathrm{j}0.47937 & 0.52327 - \mathrm{j}0.49591 \\ 0.46846 - \mathrm{j}0.64647 & 0.66929 - \mathrm{j}0.6748 & 1.3379 + \mathrm{j}1.0488 \end{bmatrix}.$$

In fact, the cubic root of matrix $\boldsymbol{A}$ should have three solutions. The other two roots can be rotated as $\boldsymbol{C}\mathrm{e}^{\mathrm{j}2\pi/3}$ and $\boldsymbol{C}\mathrm{e}^{\mathrm{j}4\pi/3}$, with the following statements

```
>> j1=exp(sqrt(-1)*2*pi/3); A1=C*j1, A2=C*j1^2,
   norm(A-A1^3), norm(A-A1^3)
```

and the other two roots, through verification, are

$$\boldsymbol{A}_1 = \begin{bmatrix} -0.9521 + \mathrm{j}0.34149 & -0.22966 + \mathrm{j}0.42961 & 0.16181 + \mathrm{j}0.29713 \\ -0.38142 + \mathrm{j}0.80579 & -1.1388 + \mathrm{j}1.0137 & 0.16784 + \mathrm{j}0.70112 \\ 0.32563 + \mathrm{j}0.72893 & 0.24974 + \mathrm{j}0.91702 & -1.5772 + \mathrm{j}0.63425 \end{bmatrix}$$

and

$$\boldsymbol{A}_2 = \begin{bmatrix} 0.18031 - \mathrm{j}0.99529 & -0.25722 - \mathrm{j}0.41369 & -0.33823 - \mathrm{j}0.008436 \\ -0.50712 - \mathrm{j}0.73321 & -0.3085 - \mathrm{j}1.4931 & -0.69111 - \mathrm{j}0.20521 \\ -0.79409 - \mathrm{j}0.082464 & -0.91904 - \mathrm{j}0.24222 & 0.23934 - \mathrm{j}1.6831 \end{bmatrix}.$$

2.2.2 Logic operations of matrices

Logical data was not implemented in earlier versions of MATLAB. The non-zero value is regarded as logic 1, while a zero value is defined as logic 0. In new versions of MATLAB, logical variables are defined and the above rules also apply.

Assume that the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are both $n \times m$ matrices, the following logical operations are defined:

(i) **"And" operation** In MATLAB, the operator `&` is used to define element-by-element "and" operation. The statement `A & B` can then be defined.

(ii) **"Or" operation** In MATLAB, the operator | is used to define element-by-element "or" operation. The statement `A | B` can then be defined.

(iii) **"Not" operation** In MATLAB, the operator ~ can be used to define the "not" operation such that `B =~A`.

(iv) **Exclusive or** The exclusive or operation of two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ can be evaluated from `xor(A, B)`.

2.2.3 Relationship operations of matrices

Various relationship operators are provided in MATLAB. For example, `C=A > B` will perform element-by-element comparison, with the element $c_{ij} = 1$ for $a_{ij} > b_{ij}$, and $c_{ij} = 0$ otherwise. The equality relationship can be tested with `==` operator, while the other operators `>=`, `~=` can also be used.

The special functions such as `find()` and `all()` can also be used to perform relationship operations. For instance, the index of the elements in $\boldsymbol{C}$ equal to 1 can be obtained from `find(C==1)`. The following commands can be used:

```
>> A=[1,2,3; 4 5,6; 7,8 0]; % enter a matrix
   i=find(A>=5)' % find all the indices in A whose value is larger than 5
```

and the indices can be found as $i = 3, 5, 6, 8$. It can be seen that the function arranges first the original matrix $\boldsymbol{A}$ in a single column, on a columnwise basis. The indices can then be returned.

The functions `all()` and `any()` can also be used to check the values in the given matrices. For instance

```
>> a1=all(A>=5) % check each column whether all larger than 5
   a2=any(A>=5) % check each column whether any larger than 5
```

and it can be found that $\boldsymbol{a}_1 = [0, 0, 0]$, $\boldsymbol{a}_2 = [1, 1, 1]$.

2.2.4 Simplifications and presentations of analytical results

The Symbolic Math Toolbox can be used to derive mathematical formulas. The results however are often not presented in their simplest form. The results should then be simplified. The easiest way of simplification is by the use of `simple()` function, where different simplification methods are tested automatically until the simplest result can be obtained, with the syntaxes

```
s1=simple(s), % try various simplification methods and find the simplest
[s1,how]=simple(s), % return the simplest form and the method used
```

where s is the original expression, and s_1 is the simplified result. The string argument `how` will return the method of simplification. Apart from the easy-to-use `simple()` function, the function `collect()` can be used to collect the coefficients, and function `expand()` can be used to expand a polynomial. The function `factor()` can be used to perform factorization of a polynomial. The function `numden()` can be used to extract the numerator and denominator from a given expression.

Example 2.7 If a polynomial $P(s)$ is given by $P(s) = (s+3)^2(s^2+3s+2)(s^3+12s^2+48s+64)$, process it with various functions and understand the results converted.

Solution A symbolic variable s should be declared first, then the full polynomial can be expressed easily and the polynomial can then be established in MATLAB workspace. With the polynomial, one can first simplify it with the `simple()` function

```
>> syms s; P=(s+3)^2*(s^2+3*s+2)*(s^3+12*s^2+48*s+64)
   [P1,m]=simple(P) % a series of simplications made, find the simplest
```

and one finds that $P_1 = (s+3)^2(s+2)(s+1)(s+4)^3$, with the method m=`factor`, which means that factorization method is used to reach the conclusion. Also the `expand()` function can be tested

```
>> expand(P)              % expand the polynomial
```

and the expanded polynomial is $s^7+21s^6+185s^5+883s^4+2454s^3+3944s^2+3360s+1152$.

The function `subs()` provided in the Symbolic Math Toolbox can be used to perform variable substitution, and the syntaxes are

f_1=`subs(`f,x_1,x_1^*`)` or f_1=`subs(`f,$\{x_1, x_2, \cdots, x_n\}$,$\{x_1^*, x_2^*, \cdots, x_n^*\}$`)`

where f is the original expression. With the statement, the variable x_1 in the original function can be substituted with a new variable or expression x_1^*. The result is given in the variable f_1. The latter syntax can be used to substitute many variables simultaneously.

The function `latex()` can be used to convert a symbolic expression into a LaTeX-readable string, which can be embedded into a LaTeX document.

Example 2.8 For a given function $f(t) = \cos(at+b) + \sin(ct)\sin(dt)$, evaluate its Taylor expression with the function `taylor()` and convert the results in LaTeX.

Solution A full description on Taylor series expansion will be given in Section 3.2. Here the function `taylor()` can be used straightforwardly to get the results. Applying the function `latex()` to the results, the LaTeX can be obtained.

```
>> syms a b c d t;                  % declare symbolic variables
   f=cos(a*t+b)+sin(c*t)*sin(d*t); % define the function f(t) with taylor()
   f1=taylor(f,5);                  % find first 5 terms in Taylor series
   latex(f1)                        % can be converted to a LaTeX string
```

The results can be embedded into a LaTeX document, and through compilation, the following results can be obtained

$$f(x) \approx \cos b - at\sin b + \left(-\frac{a^2\cos b}{2} + cd\right)t^2 + \frac{a^3\sin b}{6}t^3 + \left(\frac{a^4\cos b}{24} - \frac{cd^3}{6} - \frac{c^3d}{6}\right)t^4.$$

Unfortunately, there are no directly usable converters to other word processing programs such as Microsoft Word.

2.2.5 Basic number theory computations

Basic data transformation and number theory functions are provided in MATLAB, as shown in Table 2.1. The following examples are used to illustrate the functions. Through the example, the readers can observe the results.

TABLE 2.1: Functions for data transformations

function	syntax	function description
`floor()`	$\boldsymbol{n}$=`floor(`$\boldsymbol{x}$`)`	round towards $-\infty$ for each value in variable $\boldsymbol{x}$, mathematically denoted as $\boldsymbol{n} = [\boldsymbol{x}]$
`ceil()`	$\boldsymbol{n}$=`ceil(`$\boldsymbol{x}$`)`	round towards $+\infty$ for $\boldsymbol{x}$
`round()`	$\boldsymbol{n}$=`round(`$\boldsymbol{x}$`)`	round to nearest integer for $\boldsymbol{x}$
`fix()`	$\boldsymbol{n}$=`fix(`$\boldsymbol{x}$`)`	round towards zero for variable $\boldsymbol{x}$
`rat()`	`[`$\boldsymbol{n}$`,`$\boldsymbol{d}$`]=rat(`$\boldsymbol{x}$`)`	find rational approximation for variable $\boldsymbol{x}$, and the numerator and denominator are returned respectively in $\boldsymbol{n}$ and $\boldsymbol{d}$
`rem()`	$\boldsymbol{B}$=`rem(`$\boldsymbol{A}$`,`$\boldsymbol{C}$`)`	find the reminder after division to variable $\boldsymbol{A}$
`gcd()`	k=`gcd(`n`,`m`)`	compute the greatest common divisor for n and m
`lcm()`	k=`lcm(`n`,`m`)`	compute the least common multiplier for n and m
`factor()`	`factor(`n`)`	prime factorization
`isprime()`	$\boldsymbol{v}_1$=`isprime(`$\boldsymbol{v}$`)`	check whether each component in $\boldsymbol{v}$ is prime or not. Set the corresponding value in $\boldsymbol{v}_1$ to 1 for prime numbers, otherwise set to 0

Example 2.9 For a given data set $-0.2765, 0.5772, 1.4597, 2.1091, 1.191, -1.6187$, observe the integers obtained using different rounding functions.

Solution The following statements can be used to round the original vector such that

```
>> A=[-0.2765,0.5772,1.4597,2.1091,1.191,-1.6187];
   v1=floor(A), v2=ceil(A) % round towards -∞ and +∞ respectively
   v3=round(A), v4=fix(A)  % round towards 0 and nearest integers
```

and the integer vectors obtained are $\boldsymbol{v}_1 = [-1, 0, 1, 2, 1, -2]$, $\boldsymbol{v}_2 = [0, 1, 2, 3, 2, -1]$, $\boldsymbol{v}_3 = [0, 1, 1, 2, 1, -2]$, $\boldsymbol{v}_4 = [0, 0, 1, 2, 1, -1]$.

Example 2.10 Assume that a 3×3 Hilbert matrix can be specified with the statement $\boldsymbol{A}$=`hilb(3)`, perform the rational transformation.

Solution The following statements can be used to find the rational approximation

```
>> A=hilb(3); [n,d]=rat(A)
```

and the integer matrices obtained are $\boldsymbol{n} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, $\boldsymbol{d} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$.

Example 2.11 Find the greatest common divisor and least common multiplier to the numbers 1856120 and 1483720, and find the prime factorization to the least common multiplier obtained.

Solution Since the values are very large, one should not use the double-precision representations. The symbolic representations must be used instead. The following statements can be used

```
>> m=sym(1856120); n=sym(1483720);
   gcd(m,n), lcm(m,n), factor(lcm(n,m))
```

which yield the greatest common divisor of 1960 and the greatest common multiplier of 1405082840, whose prime factorization is $(2)^3(5)(7)^2(757)(947)$.

Here the functions `gcd()` and `lcm()` can only be used to deal with two variables. If more than two variables are expected, nested calls are allowed such that `gcd(gcd(`m`,`n`),`k`)`.

Example 2.12 List all the prime numbers in the interval $[1, 1000]$.

Solution The prime numbers can easily be recognized by the function `isprime(`$\boldsymbol{A}$`)`. All the prime numbers less than 1000 can be extracted as shown in Table 2.2.

```
>> A=1:1000; B=A(isprime(A))
```

TABLE 2.2: The prime numbers less than 1000

2	29	67	107	157	199	257	311	367	421	467	541	599	647	709	769	829	887	967
3	31	71	109	163	211	263	313	373	431	479	547	601	653	719	773	839	907	971
5	37	73	113	167	223	269	317	379	433	487	557	607	659	727	787	853	911	977
7	41	79	127	173	227	271	331	383	439	491	563	613	661	733	797	857	919	983
11	43	83	131	179	229	277	337	389	443	499	569	617	673	739	809	859	929	991
13	47	89	137	181	233	281	347	397	449	503	571	619	677	743	811	863	937	997
17	53	97	139	191	239	283	349	401	457	509	577	631	683	751	821	877	941	
19	59	101	149	193	241	293	353	409	461	521	587	641	691	757	823	881	947	
23	61	103	151	197	251	307	359	419	463	523	593	643	701	761	827	883	953	

2.3 Flow Control Structures of MATLAB Language

As a programming language, the loop control structures, conditional control structures, switch structures and trial structures are provided in MATLAB. These structures are illustrated in this section.

2.3.1 Loop control structures

The loop structures can be introduced by the keywords `for` or `while`, and ended with the `end` command. The two kinds of loop structures are shown in Figures 2.1 (a) and (b), respectively.

(i) **The `for` loop structures** The syntax of the structure is

`for` $i=\boldsymbol{v}$`,` *loop programs body*`,` `end`

When using the `for` loop structure, a component in vector $\boldsymbol{v}$ is extracted and assigned to variable i each time, the loop body can be executed. Then go back to the `for` statement, until all the components in $\boldsymbol{v}$ are used.

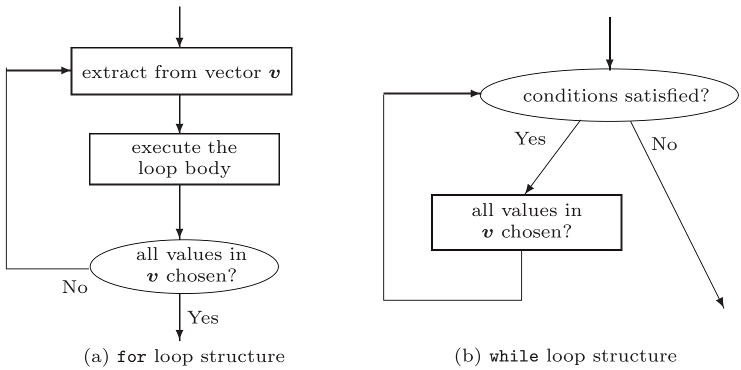


FIGURE 2.1: Illustration of loop structures

(ii) **The `while` loop structures** The syntax of the structure is

`while (`*condition*`),` *loop structure body*`,` `end`

The *condition* expression is crucial in the `while` loop structure. If it is true, the loop will be executed, and returned back to the `while` command. The loop structure will be executed, until *condition* becomes false.

There are differences between the two functions. Examples will be given below to show the advantages and disadvantages of these structures.

Example 2.13 Compute the sum of $\sum_{i=1}^{100} i$ using loop structures.

Solution The `for` and `while` loop structures can be used with the following statements, and the same results can be obtained

```
>> s1=0; for i=1:100, s1=s1+i; end
   s2=0; i=1; while (i<=100), s2=s2+i; i=i+1; end; s1, s2
```

where it can be seen that the `for` loop structure is simpler. In fact, the simplest statement for this problem is `sum(1:100)`. In the function call, the built-in function `sum()` can be used to solve the problem.

Example 2.14 Find the minimum value of m such that $\sum_{i=1}^{m} i > 10000$.

Solution It can be seen that it is not possible to solve such a problem with the `for` loop structure. However, the structure of `while` can be used easily to find the value of m

```
>> s=0; m=0;
   while (s<=10000), m=m+1; s=s+m; end, [s,m] % the value of m is expected
```

with $m = 141$ and the sum is $s = 10011$.

The loop statements can be used in nested format. The statement `break` can be used to terminate the loop structure of the current level.

The speed of loop is slow in MATLAB, compared with other programming languages. Thus the loops should be avoided, and vectorized programming techniques should be used instead.

Example 2.15 Evaluate the sum of the series[1] $S = \sum_{i=1}^{100000} \left(\frac{1}{2^i} + \frac{1}{3^i}\right)$.

Solution The execution time can be measured with the statements `tic` and `toc`. The time needed in vectorization is about 0.116 seconds, and the one needed in loops is 0.443 seconds. Thus the vectorization method is normally faster.

```
>> tic, s=0; for i=1:100000, s=s+1/2^i+1/3^i; end; toc
   tic, i=1:100000; s=sum(1./2.^i+1./3.^i); toc
```

2.3.2 Conditional control structures

Conditional control structures are the most widely used control structures. In MATLAB, `if ··· end` structure, as well as the complicated ones with `else` and `elseif` can be used. The structures can be shown in Figure 2.2.

```
if (condition 1) % If condition 1 is satisfied, statement group 1 is executed.
      statement group 1   % other sub-level if can be nested
elseif (condition 2) % Otherwise, if condition 2 is met, group 2 is executed.
      statement group 2
          ⋮   ⋮        % more conditional control statements
else    % if none of the above conditions are satisfied, define defaults.
      statement group n+1
end
```

[1]In order to demonstrate the performance of vectorization, the number of terms are exaggerated. Normally 20∼30 terms will be adequate for the exact solutions. The performance of loops was speeded up in new versions of MATLAB. And in version 7.x, the speed of loop execution is close to the vectorization method.

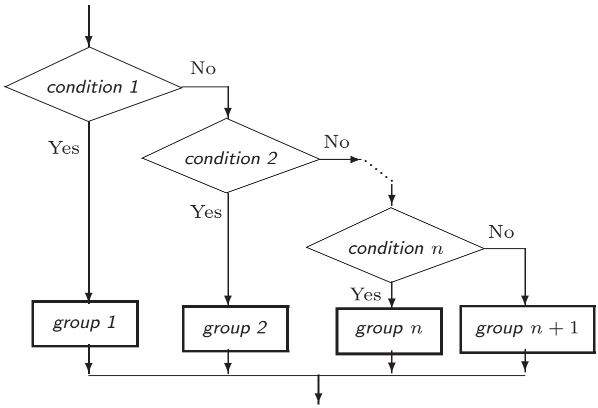


FIGURE 2.2: Illustrations of conditional control structures

Example 2.16 Solve the problem in Example 2.14 again using `for` and `if` statements.

Solution It has been shown in Example 2.14 that the `for` loop structure is not suitable for finding the minimum m such that the sum is greater than 10000. The `for` loop can be used with `if` structure to solve the problem.

```
>> s=0;
   for i=1:10000, s=s+i; if s>10000, break; end, end, s
```

Thus the structure of the program is more complicated than that of the `while` structure.

2.3.3 Switch structure

The switch structure is illustrated in Figure 2.3, and the fundamental structure is

```
switch switch expression
case expression 1, statements 1
case {expression 2, expression 3,···, expression m}, statements 2
      ⋮
otherwise, statements n
end
```

where the crucial part in switch structure is the evaluation of *switch expressions*. If it matches a value in a case statement, the statements after the case statement should be executed. Once completed, the switch structure is terminated.

There exist differences between the switch statements in MATLAB and in C languages. The following tips should be noted in programming with MATLAB:

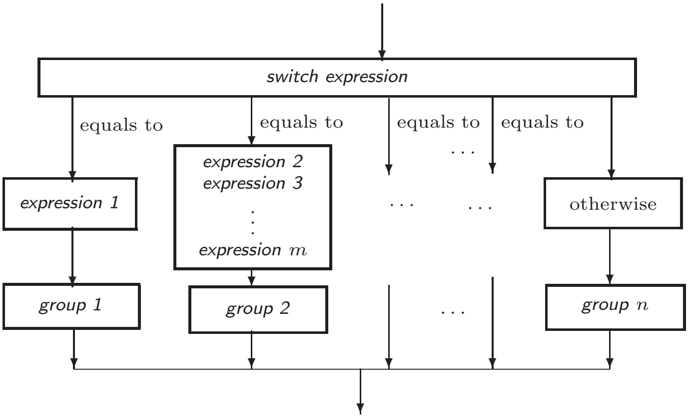


FIGURE 2.3: Illustrations of switch structures

(i) When the value of the switch expression equals *expression 1*, the *statement group 1* should be executed. After execution, the structure is completed. There is no need to introduce a `break` statement before the next `case`.

(ii) If one is checking whether one of several expressions is satisfied, the expressions must be given in cell format.

(iii) If none of the expressions are satisfied, the paragraph in `otherwise` should be executed. It is similar to the `default` statement in C language.

(iv) The execution results are independent of the orders of the `case` statement. When there exist two or more `case` statements having the same expressions, those listed behind may never be executed.

2.3.4 Trial structure

A brand new trial structure is provided in MATLAB, whose syntax is

```
try,    statement group 1,
catch,    statement group 2,
end
```

Normally, only the *statement group 1* is executed. However, if an error occurs during execution of any of the statements, the error is captured into `lasterror`, and the *statement group 2* is executed. The new structure is not available in languages such as C. The trial structure is useful in practical programming.

2.4 Writing and Debugging MATLAB Functions

Two types of source programs are supported in MATLAB, both in ASCII format. One of the code is the M-script program, which is a series of MATLAB statements to be evaluated in sequence, just as the batch files in DOS. The execution of this type of program is simple, one can simply key in the file name under the >> prompt. M-scripts process the data in MATLAB workspace, and the results are returned back to MATLAB workspace. M-scripts are suitable for dealing with small-scale computations.

Example 2.17 Consider again the problem in Example 2.14. The program can be used to find the smallest m such that the summation is greater than 10000. If one wants to find such m's for the summation greater than 20000 or 30000, the original program should be modified. This method is quite complicated and inconvenient. If a mechanism can be established such that the user may define 20000 or 30000 externally, without modifying the original program, the mechanism is quite reasonable. This kind of mechanism is often referred to as the *functions*.

M-function is the major structure in MATLAB programming. In practical programming, M-script programming is not recommended. In this section, MATLAB functions and some tricks in programming are given.

2.4.1 Basic structure of MATLAB functions

MATLAB functions are led by the statement of `function`, whose basic structure is

```
function [return argument list] =funname(input argument list)
    comments led by % sign
    input and output variables check
    main body of the function
```

The actual numbers of input and returned arguments can be extracted respectively by `nargin` and `nargout`. In the function call, the two variables are generated automatically.

If more than one input or returned arguments are needed, they should be separated with commas in the lists. The comments led by `%` will not be executed. The messages in the leading comments can be displayed by the `help` command.

From the system view points, the MATLAB functions can be regarded as a variable processing unit, which receives variables from the calling function. Once the variables are processed, the results will be returned back to the calling function. Apart from the input and returned arguments, the other variables within the function are local variables, which will be lost after function calls. Examples will be given to demonstrate the programming techniques.

Example 2.18 Consider the requests in Example 2.17. One may choose the input argument as k, and returned arguments of m and s, where s is the sum of first m terms. The function can then be written as

```
function [m,s]=findsum(k)
s=0; m=0; while (s<=k), m=m+1; s=s+m; end
```

The previous function can be saved as a function `findsum.m`. One can then call such a function for different values of k, without modifying the function. For instance, if the targeted summation is 145323, the following statements can be used to find the smallest value of m, which returns $m = 539$, $s_1 = 145530$.

```
>> [m1,s1]=findsum(145323)
```

It can be seen that the calling format is quite flexible, and we may find the needed results without modifying the original program. Thus this kind of method is recommended in programming.

Example 2.19 Assume that a MATLAB function is needed in obtaining an $n \times m$ Hilbert matrix[2], whose (i, j)th element is $h_{i,j} = 1/(i + j - 1)$. The following additional requests are also to be implemented:

(i) If only one input argument n is given in the calling command, a square matrix should be generated, such that $m = n$.
(ii) Certain help information to this function is required.
(iii) Check the formats of input and returned arguments.

Solution In actual programming, it is better to write adequate comments, which are beneficial to the programmer as well as to the maintainer of the program. The required MATLAB function `myhilb()` can be written and stored as `myhilb.m` in the default MATLAB path.

```
function A=myhilb(n, m)
%MYHILB  The function is used to illustrate MATLAB functions.
%   A=MYHILB(N, M) generates an NxM Hilbert matrix A;
%   A=MYHILB(N) generates an NxN square Hilbert matrix A;
%
%See also: HILB.

%  Designed by Professor Dingyu XUE, Northeastern University, PRC
%     5 April, 1995, Last modified by DYX at 30 July, 2001
if nargout>1, error('Too many output arguments.'); end
if nargin==1, m=n;   % if one input argument used, square matrix
elseif nargin==0 | nargin>2
   error('Wrong number of iutput arguments.');
end
```

[2] A function `hilb()` is provided in MATLAB to create an $n \times n$ square Hilbert matrix.

```
for i=1:n, for j=1:m, A(i,j)=1/(i+j-1); end, end
```

In the program, the comments are led by % sign. To implement the requirement in item (i), one should check whether the number of input argument is 1, i.e., whether `nargin` is 1. If so, the column number m is set to n, the row number, thus a square matrix can be generated. If the numbers of input or returned arguments are not correct, the error messages can be given. The double `for` loops will generate the required Hilbert matrix.

The on-line help command `help myhilb` will display the following information

```
MYHILB  The function is used to illustrate MATLAB functions.
   A=MYHILB(N, M) generates an NxM Hilbert matrix A;
   A=MYHILB(N) generates an NxN square Hilbert matrix A;
See also: HILB.
```

It should be noted that only the first few lines of information are displayed, while the author information is not displayed. This is because there is a blank line before the author information.

The following commands can be used to generate Hilbert matrices

```
>> A1=myhilb(4,3)   % two input arguments yield a rectangular matrix
   A2=myhilb(4)     % while one input argument yields a square matrix
```

and the two matrices can then be established as

$$\boldsymbol{A}_1=\begin{bmatrix}1 & 0.5 & 0.33333\\ 0.5 & 0.33333 & 0.25\\ 0.33333 & 0.25 & 0.2\\ 0.25 & 0.2 & 0.16667\end{bmatrix},\quad \boldsymbol{A}_2=\begin{bmatrix}1 & 0.5 & 0.33333 & 0.25\\ 0.5 & 0.33333 & 0.25 & 0.2\\ 0.33333 & 0.25 & 0.2 & 0.16667\\ 0.25 & 0.2 & 0.16667 & 0.14286\end{bmatrix}.$$

Example 2.20 MATLAB functions can be called recursively, i.e., a function may call itself. Please write a recursive function to evaluate the factorial $n!$.

Solution Consider the factorial $n!$. From the definition $n! = n(n-1)!$, it can be seen that the factorial of n can be evaluated from the factorial of $n-1$, while $n-1$ can be evaluated from $n-2$, and so on. The exits of the function call should be $1! = 0! = 1$. Thus the recursive function can be written as follows, with the comments omitted.

```
function k=my_fact(n)
if nargin~=1, error('Error: Only one input variable accepted'); end
if abs(n-floor(n))>eps | n<0 % judge whether n is a non-negative integer
   error('n should be a non-negative integer');
end
if n>1       % if n > 1, recursive calls are used
   k=n*my_fact(n-1);
elseif any([0 1]==n) % 0! = 1! = 1, the exit of the function
   k=1;
end
```

It can be seen that, in the function, the judgement whether n is a non-negative integer is made. If not, an error message will be declared. If it is, the recursive function calls will be used such that when $n = 1$ or 0, the result is 1, which can be used as an exit to the function. For instance, 11! can be evaluated with `my_fact(11)`, and the result obtained is 39916800.

In fact, the factorial for any non-negative integer can be evaluated directly with function `factorial(`n`)`, and the kernel of such a function is `prod(1:`n`)`.

Example 2.21 Compare the advantages and disadvantages of recursive algorithm with loop structure in constructing the Fibonacci arrays.

Solution It is for sure that the recursive algorithm is an effective method for a class of problems. However, this method should not be misused. A counter-example is shown in this example. Consider the Fibonacci array, where $a_1 = a_2 = 1$, and the kth term can be evaluated from $a_k = a_{k-1} + a_{k-2}$ for $k = 3, 4, \cdots$. A MATLAB function can be written for the problem

```
function a=my_fibo(k)
if k==1 | k==2, a=1; else, a=my_fibo(k-1)+my_fibo(k-2); end
```

and for $k = 1, 2$, the exit can be made such that it returns 1. If the 25th term is expected, the following statements can be used and the time required is 7.6 seconds.

```
>> tic, my_fibo(25), toc
```

If one is expecting the term $k = 30$, several hours of time might be required. If the loop structure is used, within 0.02 second, the whole array can be obtained for $k = 100$.

```
>> tic, a=[1,1]; for k=3:100, a(k)=a(k-1)+a(k-2); end, toc
```

It can be seen that the ordinary loop structure only requires a very short execution time. Thus the recursive function call should not be misused.

2.4.2 Programming of functions with variable inputs/outputs

In the following presentation, the variable number of input and returned arguments is introduced, based on the cell data type. It should be mentioned that most of the MATLAB functions are implemented in this format.

Example 2.22 The product of two polynomials can be evaluated from the `conv()` function, based on the algorithm of finding the convolution of two arrays. Write a function to evaluate directly the multiplications of arbitrary number of polynomials.

Solution Cell data type can be used to write the function `convs()`, which can be used to evaluate the multiplication of arbitrary number of polynomials.

```
function a=convs(varargin)
a=1; for i=1:length(varargin), a=conv(a,varargin{i}); end
```

The input argument list is passed to the function through the cell variable `varargin`. Consequently, the returned arguments can be specified in `varargout`, if necessary. Under such a function, the multiplication of arbitrary number of polynomials can be obtained. The following statements can be used to call the function

```
>> P=[1 2 4 0 5]; Q=[1 2]; F=[1 2 3]; D=convs(P,Q,F)
   E=conv(conv(P,Q),F)  % nested calls are to be used with conv() function
   G=convs(P,Q,F,[1,1],[1,3],[1,1])
```

where the obtained vectors are respectively

$\boldsymbol{E} = [1, 6, 19, 36, 45, 44, 35, 30]^{\mathrm{T}}$, $\boldsymbol{G} = [1, 11, 56, 176, 376, 578, 678, 648, 527, 315, 90]^{\mathrm{T}}$.

2.4.3 Inline functions and anonymous functions

In order to describe simply the mathematics functions, *inline functions* can be used. The functions are equivalent to the M-functions. However, with inline function, it may no longer be necessary to save files. The format of inline function is `fun=inline(function expression, list of variables)`, where the *function expression* is the actual contents of the function to be expressed, and the *list of variables* contains all the input variables, with each variable given as a string. The inline function is useful in the descriptions in differential equations and objective function given later. The function type accept only one returned variable. The mathematical function $f(x, y) = \sin(x^2 + y^2)$ can be expressed as `f=inline('sin(x.^2+y.^2)','x','y')`.

Anonymous function provided in MATLAB 7.x is a brand new type of function definition, the structure of the function is similar to the inline function, but it is more concise and easy to use. The syntax of the function is

`f=@(list of variables) function_contents`, e.g., `f=@(x,y)sin(x.^2+y.^2)`

Note that the variable currently existing in MATLAB workspace can be used directly in the function. For instance, the variables a and b in MATLAB workspace can be used in the anonymous function

```
f=@(x,y)a*x.^2+b*y.^2
```

to describe the mathematical function $f(x, y) = ax^2 + by^2$. If such a function has been defined, while the variables a, b change after that, the values of those in the anonymous function will not change, unless it is defined again.

2.5 Two-Dimensional Graphics

Graphics and visualization are the most significant advantages of MATLAB. A series of straightforward and simple functions are provided in MATLAB for two-dimensional and three-dimensional graphics. Experimental and simulation results can be easily interpreted in graphical form. In this section, the two-dimensional graphics functions will be illustrated.

2.5.1 Basic statements of two-dimensional plotting

Assume that a sequence of experimental data is acquired. For instance, at time instances $t = t_1, t_2, \cdots, t_n$, the function values are $y = y_1, y_2, \cdots, y_n$. The data can be entered to MATLAB workspace such that $\boldsymbol{t} = [t_1, t_2, \cdots, t_n]$ and $\boldsymbol{y} = [y_1, y_2, \cdots, y_n]$. The command `plot(t,y)` can be used to draw the curve for the data. The "curve" is in fact represented by poly-lines, joining the sample points.

It can be seen that the syntax of the function is quite straightforward. In actual applications, the `plot()` function can also be called in other extended ways.

(i) $\boldsymbol{t}$ is still a vector and $\boldsymbol{y}$ can be expressed by a matrix such that

$$\boldsymbol{y} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix}.$$

The same function can also be used to draw m curves, with each row of matrix $\boldsymbol{y}$ corresponding to a curve.

(ii) $\boldsymbol{t}$ and $\boldsymbol{y}$ are both matrices, and the sizes of the two matrices are the same. The plots between each row of $\boldsymbol{t}$ and $\boldsymbol{y}$ can be drawn.

(iii) Assume that there are many pairs of such vectors or matrices, ($\boldsymbol{t}_1$, $\boldsymbol{y}_1$), ($\boldsymbol{t}_2$, $\boldsymbol{y}_2$), $\cdots$, ($\boldsymbol{t}_m$, $\boldsymbol{y}_m$), the following statement can be used directly to draw the corresponding curves.

`plot(`$\boldsymbol{t}_1, \boldsymbol{y}_1, \boldsymbol{t}_2, \boldsymbol{y}_2, \cdots, \boldsymbol{t}_m, \boldsymbol{y}_m$`)`

(iv) The line types, line width and color information of the curves can separately be specified with the command

`plot(`$\boldsymbol{t}_1, \boldsymbol{y}_1,$ *option 1*, $\boldsymbol{t}_2, \boldsymbol{y}_2,$ *option 2*, $\cdots$, $\boldsymbol{t}_m, \boldsymbol{y}_m,$ *option* m`)`

where the available *options* are shown in Table 2.3. The combinations of the options are also allowed. For instance, the combination `'r-.pentagram'` indicates the red dash dot curve, with the sampling points marked by pentagrams.

After the curves are drawn, the command `grid on` can be used to add grids to the curves, while the `grid off` command may remove the grids. Also `hold on` command can reserve the current current axis. Other `plot()` function can be used to superimpose curves on top of the existing ones. `hold off` command may remove the holding status.

Example 2.23 Draw the curve of $y = \sin(\tan x) - \tan(\sin x)$ in the interval $x \in [-\pi, \pi]$.

Solution The curve of $f(x)$ can be drawn easily with the following statements

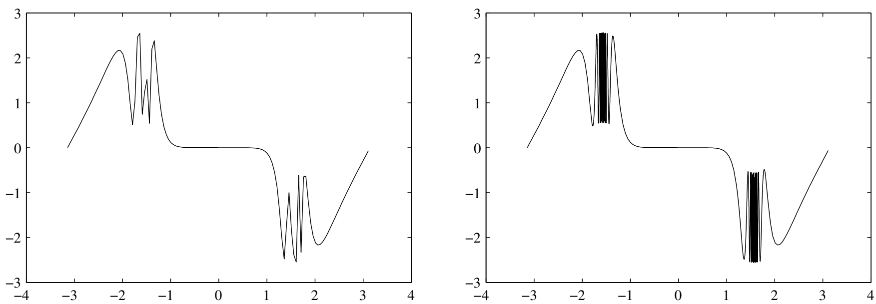
```
>> x=[-pi : 0.05: pi];          % specify the vector with a step-size of 0.05
```

TABLE 2.3: Options in MATLAB plotting commands

line type		line color				markers			
opts	meaning	opts	meaning	opts	meaning	opts	meaning	opts	meaning
'-'	solid	'b'	blue	'c'	cyan	'*'	$*$	'pentagram'	☆
'--'	dash	'g'	green	'k'	black	'.'	dotted	'o'	○
':'	dotted	'm'	magenta	'r'	red	'x'	$\times$	'square'	□
'-.'	dash-dot	'w'	white	'y'	yellow	'v'	$\bigtriangledown$	'diamond'	◇
'none'	none					'^'	$\bigtriangleup$	'hexagram'	✡
						'>'	$\triangleright$	'<'	$\triangleleft$

```
y=sin(tan(x))-tan(sin(x)); % evaluate the function values
plot(x,y)                  % draw the curve
```

and the curve in Figure 2.4 (a) can be obtained.



(a) with the default step-size of 0.05 (b) improved curve with variable-step-sizes

FIGURE 2.4: Two dimension curve for the given function

It can be seen from the curve that it is rather sluggish over the intervals $x \in (-1.8, -1.2)$ and $x \in (1.2, 1.8)$, since the step-size 0.05 is too large for these intervals. The step-size for these intervals should be selected smaller such that

```
>> x=[-pi:0.05:-1.8,-1.801:.001:-1.2, -1.2:0.05:1.2,...
      1.201:0.001:1.8, 1.81:0.05:pi]; % with variable-step-size
   y=sin(tan(x))-tan(sin(x));         % evaluate the function
   plot(x,y)                          % draw the curve
```

The modified curve of the function is given in Figure 2.4 (b). It can be seen that the curve is significantly improved in the new plot. Alternatively, for the whole interval, a fixed step-size of 0.001 can be selected.

Example 2.24 Please draw the saturation function $y = \begin{cases} 1.1\mathrm{sign}(x), & |x| > 1.1 \\ x, & |x| \leqslant 1.1 \end{cases}$.

Solution It is obvious that one can create a vector of $\boldsymbol{x}$, then for each point, construct an `if` clause to calculate the value of $\boldsymbol{y}$. An alternative way is to use vectorized format to evaluate the function values. With the following statements, the segmented function can be drawn as shown in Figure 2.5.

```
>> x=[-2:0.02:2];  % generate the x vector
   y=1.1*sign(x).*(abs(x)>1.1) + x.*(abs(x)<=1.1); plot(x,y)
```

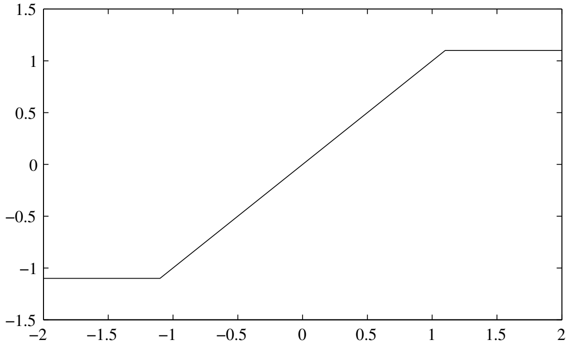


FIGURE 2.5: Segmented saturation function

Even more simply, the command `plot([-3,-1.1,1.1,3],[-1.1 -1.1 1.1 1.1])` can be used to draw the saturation poly-lines.

In MATLAB graphics, each curve or the axis is an object, and the window is another object. The properties of the objects can be assigned by `set()` function, or extracted by `get()` function. The syntaxes of the two functions are

```
set(handle,'p_name 1',p_value 1,'p_name 2',p_value 2, ···)
v=get(object, 'p_name')
```

where *p_name* and *p_value* are respectively the names and values of the corresponding properties. These two functions are very useful in graphical user interface programming.

2.5.2 Other two-dimensional plotting statements

Apart from the standard Descartes coordinate curves, MATLAB also provides other special two-dimensional graphical functions, and the common syntaxes of the functions are given in Table 2.4. In the functions, parameters $\boldsymbol{x}$, $\boldsymbol{y}$ are respectively the horizontal and vertical axis data, and `c` the color options. The parameters $\boldsymbol{y}_{\mathrm{m}}$, $\boldsymbol{y}_{\mathrm{M}}$ are the vectors of lower- and upper-boundaries in error plots. The functions are demonstrated through the following examples.

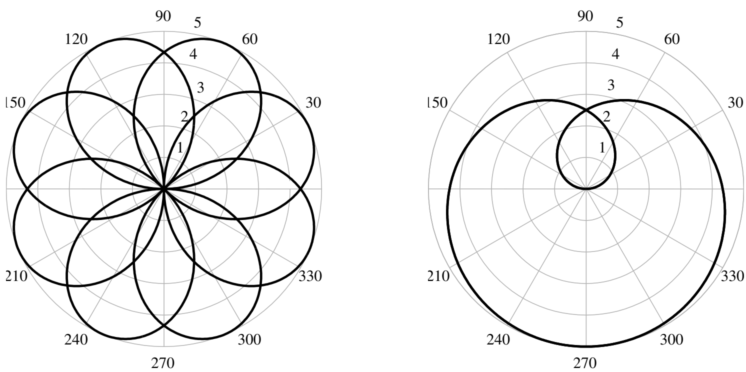
Example 2.25 Draw the polar plots for functions $\rho = 5\sin(4\theta/3)$ and $\rho = 5\sin(\theta/3)$.

Solution A vector $\boldsymbol{\theta}$ can be constructed first, over the interval $\theta \in (0, 6\pi)$, then

TABLE 2.4: Other two-dimensional plotting functions

general syntax	explanation	general syntax	explanation
`bar(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	two-dimensional bar chart	`comet(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	comet trajectory
`compass(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	compass plot	`errorbar(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`,`$\boldsymbol{y}_{\rm m}$`,`$\boldsymbol{y}_{\rm M}$`)`	errorbar plot
`feather(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	feather plot	`fill(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`,c)`	filled plot
`hist(`$\boldsymbol{y}$`,`n`)`	histogram	`loglog(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	logarithmic plot
`polar(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	polar plot	`quiver(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	quiver graph
`stairs(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	stairs plot	`stem(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	stem plot
`semilogx(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	x-semi-logarithmic plot	`semilogy(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`)`	y-semi-logarithmic plot

the function value vector $\boldsymbol{\rho}$ can be calculated. With the `polar()` function, the polar plots can be drawn as shown in Figures 2.6 (a) and (b).



(a) $\rho = 5\sin(4\theta/3)$ (b) $\rho = 5\sin(\theta/3)$

FIGURE 2.6: Polar plots

```
>> theta=0:0.01:6*pi; rho=5*sin(4*theta/3); polar(theta,rho)
   figure; rho=5*sin(theta/3); polar(theta,rho)
```

Example 2.26 Draw the sinusoidal curve with different functions in different areas of the graphics window.

Solution The following commands can be used to draw the expected curves as shown in Figure 2.7, where function `subplot(`n`,`m`,`k`)` can be used to divide the graphics window into several parts, with n, m respectively the total numbers of rows and columns, and k indicates the serial of the area.

```
>> t=0:.2:2*pi; y=sin(t);         % generate the data for plots
   subplot(2,2,1), stairs(t,y)    % partition the graphics window
   subplot(2,2,2), stem(t,y)      % stem plot in upper-right portion
   subplot(2,2,3), bar(t,y)       % bar chart in lower-left portion
   subplot(2,2,4), semilogx(t,y)  % semilogx in lower-right portion
```

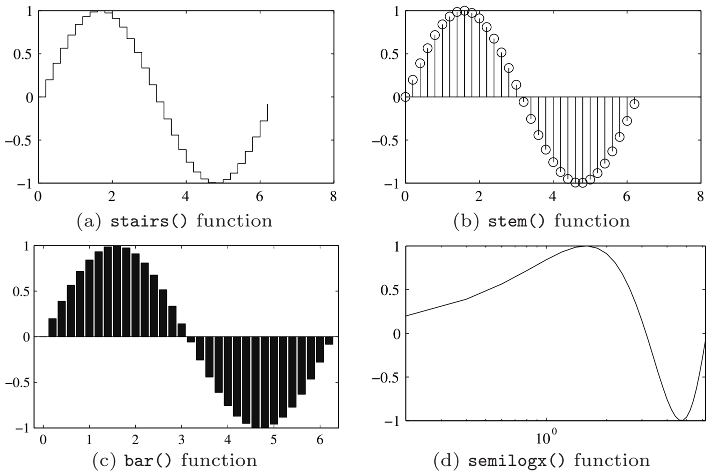


FIGURE 2.7: Different representations of the same function

2.5.3 Implicit function plotting and applications

For an implicit equation $f(x, y) = 0$, the relationship between x and y cannot be explicitly formulated. Thus the conventional `plot()` function cannot be used. The MATLAB function `ezplot()` can be used to draw the implicit function curve

`ezplot(`*implicit function expression*`)`

The following example is used to demonstrate the use of the function.

Example 2.27 Draw the curve of the implicit function

$$f(x, y) = x^2 \sin(x + y^2) + y^2 e^{x+y} + 5 \cos(x^2 + y) = 0.$$

Solution From the given function, it can be seen that the analytical explicit solution of x-y relationship cannot be found. Thus the `plot()` function cannot be used for such a function. The following MATLAB statements can be used to draw the implicit function as shown in Figure 2.8 (a).

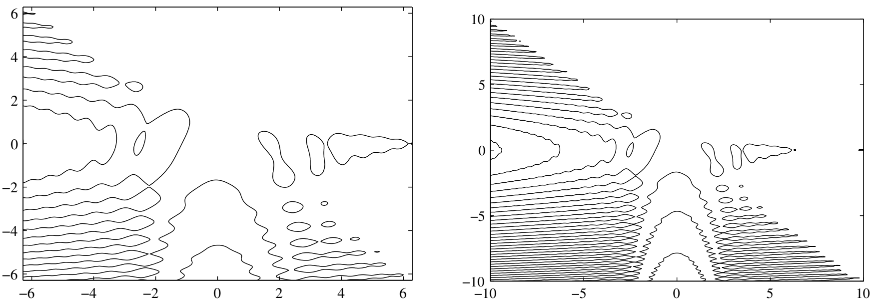
```
>> ezplot('x^2 *sin(x+y^2) +y^2*exp(x+y)+5*cos(x^2+y)')
```

The above functions selected automatically the x range, the range can be enlarged with the following statements, with the implicit curve shown in Figure 2.8 (b).

```
>> ezplot('x^2 *sin(x+y^2) +y^2*exp(x+y)+5*cos(x^2+y)',[-10 10])
```

2.5.4 Graphics decorations

The graphics window with editing tools is shown in Figure 2.9. The user may choose to apply text and arrows to the plots. Local zooming and 3D view point settings are also provided in the plots. For instance, a subset of LaTeX commands can be used to add mathematical formula to the plots.



(a) default curves (b) curves within larger area

FIGURE 2.8: Curves of implicit functions

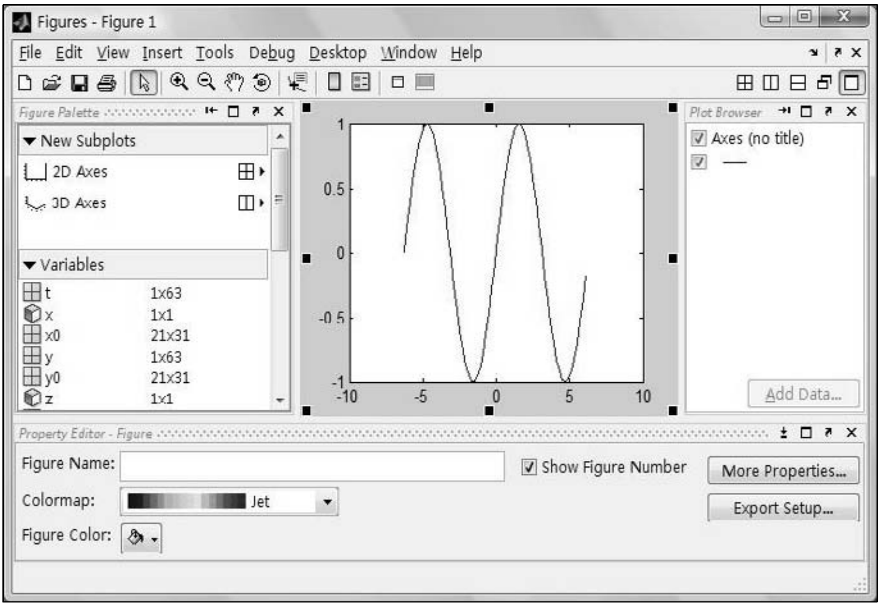


FIGURE 2.9: MATLAB graphics window with editing tools

In the graphics editing interface, there are three parts, with the left part corresponding to the View → Figure Palette menu item, where arrows and text can be added to the curve. 2D and 3D axes can also be added to the curve. The bottom part of the window corresponds to the Property Editor menu item, which allows the selections of color, line styles or fonts to the selected objects. The right part of the window corresponds to the View → Plot Browser menu item, which allows the user to add new data or superimpose new curves.

An example of a typical graphics display under the view-point change is shown in Figure 2.10, where 2D curve is displayed under a 3D framework.

LaTeX is a well established scientific type-setting system, and a subset of its mathematical symbols are supported in MATLAB. One may use them to

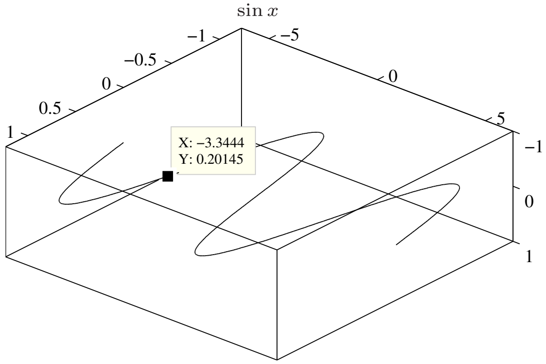


FIGURE 2.10: 3D representations of 2D curves

superimpose formula to the plots.

(i) The symbols are led by the backslash signs \, and the available symbols are listed in Table 2.5.

TABLE 2.5: TeX compatible commands in MATLAB

	c	TeX	c	TeX	c	TeX	c	TeX
lower-case Greeks	α	`\alpha`	β	`\beta`	γ	`\gamma`	δ	`\delta`
	ϵ	`\epsilon`	ε	`\varepsilon`	ζ	`\zeta`	η	`\eta`
	θ	`\theta`	ϑ	`\vartheta`	ι	`\iota`	κ	`\kappa`
	λ	`\lambda`	μ	`\mu`	ν	`\nu`	ξ	`\xi`
	o	`o`	π	`\pi`	ϖ	`\varpi`	ρ	`\rho`
	ι	`\iota`	κ	`\kappa`	ϱ	`\varrho`	σ	`\sigma`
	ς	`\varsigma`	τ	`\tau`	υ	`\upsilon`	ϕ	`\phi`
	φ	`\varphi`	χ	`\chi`	ψ	`\psi`	ω	`\omega`
upper-case Greeks	Γ	`\Gamma`	Δ	`\Delta`	Θ	`\Theta`	Λ	`\Lambda`
	Ξ	`\Xi`	Π	`\Pi`	Σ	`\Sigma`	Υ	`\Upsilon`
	Φ	`\Phi`	Ψ	`\Psi`	Ω	`\Omega`		
common maths symbols	$\aleph$	`\aleph`	$\prime$	`\prime`	$\forall$	`\forall`	$\exists$	`\exists`
	$\wp$	`\wp`	$\Re$	`\Re`	$\Im$	`\Im`	∂	`\partial`
	∞	`\infty`	∇	`\nabla`	$\surd$	`\surd`	$\angle$	`\angle`
	$\neg$	`\neg`	$\int$	`\int`	$\clubsuit$	`\clubsuit`	$\diamondsuit$	`\diamondsuit`
	$\heartsuit$	`\heartsuit`	$\spadesuit$	`\spadesuit`				
binary maths symbols	$\pm$	`\pm`	$\cdot$	`\cdot`	$\times$	`\times`	$\div$	`\div`
	$\circ$	`\circ`	$\bullet$	`\bullet`	$\cup$	`\cup`	$\cap$	`\cap`
	$\vee$	`\vee`	$\wedge$	`\wedge`	$\otimes$	`\otimes`	$\oplus$	`\oplus`
relational maths symbols	$\leqslant$	`\leq`	$\geqslant$	`\geq`	$\equiv$	`\equiv`	$\sim$	`\sim`
	$\subset$	`\subset`	$\supset$	`\supset`	$\approx$	`\approx`	$\subseteq$	`\subseteq`
	$\supseteq$	`\supseteq`	$\in$	`\in`	$\ni$	`\ni`	$\propto$	`\propto`
	$\mid$	`\mid`	$\perp$	`\perp`				
arrows	$\leftarrow$	`\leftarrow`	$\uparrow$	`\uparrow`	$\Leftarrow$	`\Leftarrow`	$\Uparrow$	`\Uparrow`
	$\rightarrow$	`\rightarrow`	$\downarrow$	`\downarrow`	$\Rightarrow$	`\Rightarrow`	$\Downarrow$	`\Downarrow`
	$\leftrightarrow$	`\leftrightarrow`	$\updownarrow$	`\updownarrow`				

(ii) Superscripts and subscripts are represented by `^` and `_` respectively. For instance, `a_2^2+b_2^2=c_2^2` represents $a_2^2+b_2^2=c_2^2$. If more than one symbol is used in the superscript, they should be written within the { and } signs. For instance `a^Abc` gives a^Abc, while `a^{Abc}` gives a^{Abc}.

LaTeX scientific type-setting system is widely used in the academic world. Interested readers may further refer to Reference [5].

2.6 Three-Dimensional Graphics

2.6.1 Plotting of three-dimensional curves

The two-dimensional function `plot()` can be extended to a three-dimensional (3D) curve drawing with the new `plot3()` function, whose syntaxes are

plot3($\boldsymbol{x}$,$\boldsymbol{y}$,$\boldsymbol{z}$)

plot3($\boldsymbol{x}_1,\boldsymbol{y}_1,\boldsymbol{z}_1$, *option 1*, $\boldsymbol{x}_2,\boldsymbol{y}_2,\boldsymbol{z}_2$, *option 2*, $\cdots$, $\boldsymbol{x}_m,\boldsymbol{y}_m,\boldsymbol{z}_m$, *option* m)

where the *options* are the same as shown in Table 2.3.

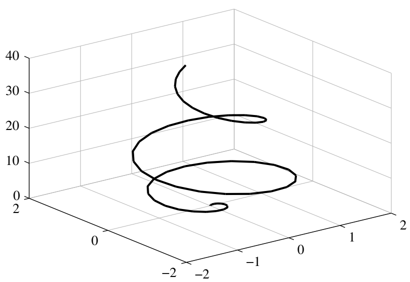
Similar to other 2D curve drawing functions, the functions `stem3()`, `fill3()` and `bar3()` can also be applied to 3D curves.

Example 2.28 Draw the curve of the parametric equations

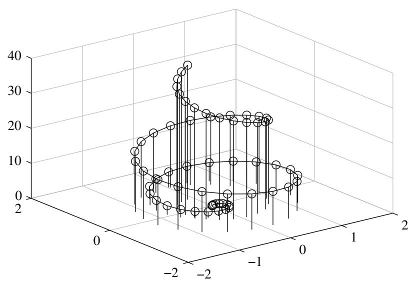
$$x(t)=t^3\sin(3t)\mathrm{e}^{-t}, y(t)=t^3\cos(3t)\mathrm{e}^{-t}, z=t^2, \quad \text{where } t\in[0,2\pi].$$

Solution A time vector $\boldsymbol{t}$ can be established first, then the vectors $\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$ can be computed. The 3D curve can be drawn with the `plot3()` function, as shown in Figure 2.11 (a). It should be noted that dot operations are used in the evaluations.

```
>> t=0:.1:2*pi;    % establish the t vector, with dot operation
   x=t.^3.*sin(3*t).*exp(-t); y=t.^3.*cos(3*t).*exp(-t); z=t.^2;
   plot3(x,y,z), grid  % 3D curve drawing
```



(a) 3D curve plots



(b) superimposed with the plot with `stem3()`

FIGURE 2.11: Three-dimensional plots

The `stem3()` function can be used to obtained the plot in Figure 2.11 (b), superimposed by the 3D curve.

```
>> stem3(x,y,z); hold on; plot3(x,y,z), grid
```

2.6.2 Plotting of three-dimensional surfaces

If function $z = f(x, y)$ is given, the 3D surface of the function can be drawn. One can generate mesh grid data in the x-y plane, with the `meshgrid()` function. The function values $\boldsymbol{z}$ can be obtained. The functions `mesh()` and `surf()` can be used to draw the 3D mesh plots and surface plots. The syntaxes of the functions are

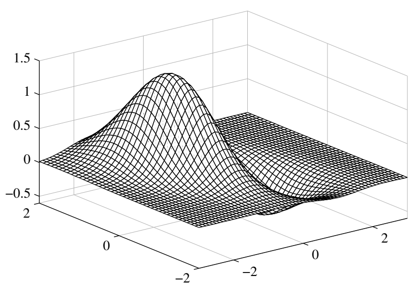
`[`$\boldsymbol{x}$`,`$\boldsymbol{y}$`]=meshgrid(`$\boldsymbol{v}_1$`,` $\boldsymbol{v}_2$`)`		% mesh grid generation
$\boldsymbol{z}$`= ...,` for instance $\boldsymbol{z}$`=`$\boldsymbol{x}$`.*`$\boldsymbol{y}$		% $\boldsymbol{z}$ matrix computation
`surf(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`,`$\boldsymbol{z}$`)` or	`mesh(`$\boldsymbol{x}$`,`$\boldsymbol{y}$`,`$\boldsymbol{z}$`)`	% mesh and surface plots

where $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are the scales in the $\boldsymbol{x}$ and $\boldsymbol{y}$ axes. The 3D surface can also be drawn with the `surfc()`, `surfl()` and `waterfall()` functions. Also the `contour()` and `contour3()` functions can be used to draw 2D and 3D contour plots.

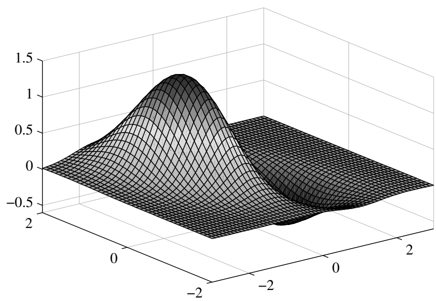
Example 2.29 Consider the function $z = f(x, y) = (x^2 - 2x)\mathrm{e}^{-x^2-y^2-xy}$. Select in the x-y plane an area and draw the 3D plots.

Solution One may use the `meshgrid()` function to specify the mesh grids on the x-y plane. The values of the function can be evaluated directly for the matrix $\boldsymbol{z}$. The mesh plot can be drawn as shown in Figure 2.12 (a).

```
>> [x,y]=meshgrid(-3:0.1:3,-2:0.1:2);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y); mesh(x,y,z)
```



(a) `mesh()` plot



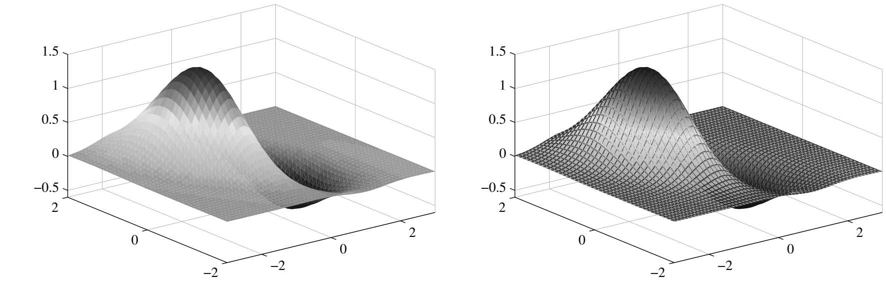
(b) `surf()` plot

FIGURE 2.12: Mesh and surface plots of a given function

If one uses `surf()` function to replace the `mesh()` function, the corresponding surface plot can be obtained as shown in Figure 2.12 (b).

```
>> surf(x,y,z)     % surface plot
```

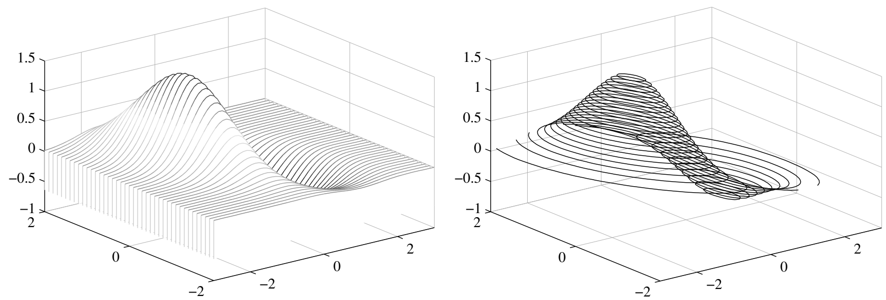
3D surface plots can be decorated by `shading` command, and the options `flat` and `interp` can be used. The decorations are shown in Figures 2.13 (a) and (b) respectively.



(a) `shading flat` (b) `shading interp`

FIGURE 2.13: 3D surfaces decorated by the `shading` command

Other functions, such as `waterfall(`$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$`)` and `contour3(`$\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}$`,30)` can be used to draw 3D plots as shown in Figures 2.14 (a) and (b).



(a) `waterfall()` plots (b) `contour3()` plots

FIGURE 2.14: Other 3D representations

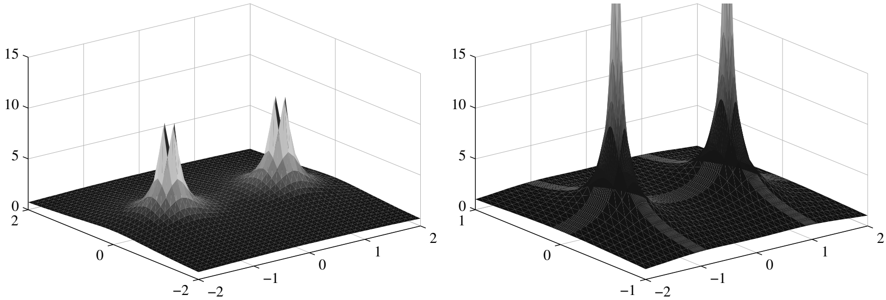
Example 2.30 Display graphically $z = f(x,y) = \dfrac{1}{\sqrt{(1-x)^2+y^2}} + \dfrac{1}{\sqrt{(1+x)^2+y^2}}$.

Solution The following statements can be used to draw the 3D surface of the function, as shown in Figure 2.15 (a).

```
>> [x,y]=meshgrid(-2:.1:2);
   z=1./(sqrt((1-x).^2+y.^2))+1./(sqrt((1+x).^2+y.^2));
   surf(x,y,z), shading flat
```

In fact, there are problems around the $(\pm 1, 0)$ points, where the function values tend to infinity. Thus variable-step-size mesh grids can be constructed, and the new 3D surface can be obtained as shown in Figure 2.15 (b).

```
>> xx=[-2:.1:-1.2,-1.1:0.02:-0.9,-0.8:0.1:0.8,0.9:0.02:1.1,1.2:0.1:2];
   yy=[-1:0.1:-0.2, -0.1:0.02:0.1, 0.2:.1:1];
   [x,y]=meshgrid(xx,yy);
```



(a) fixed-step-size (b) variable-step-size

FIGURE 2.15: Three-dimensional surfaces under different grids

```
z=1./(sqrt((1-x).^2+y.^2))+1./(sqrt((1+x).^2+y.^2));
surf(x,y,z), shading flat; zlim([0,15])
```

Example 2.31 Assume that a piecewise function is described below[6]

$$p(x_1,x_2)=\begin{cases}0.5457\exp(-0.75x_2^2-3.75x_1^2-1.5x_1), & x_1+x_2>1\\ 0.7575\exp(-x_2^2-6x_1^2), & -1<x_1+x_2\leqslant 1\\ 0.5457\exp(-0.75x_2^2-3.75x_1^2+1.5x_1), & x_1+x_2\leqslant -1.\end{cases}$$

Show the function in a three-dimensional surface.

Solution Selecting $x=x_1$ and $y=x_2$, the function value can be evaluated with the `if` statements, however the process could be very complicated. Thus the piecewise function configuration statements based on relational operations can be used to evaluate the functions as follows

```
>> [x,y]=meshgrid(-1.5:.1:1.5,-2:.1:2);
   z= 0.5457*exp(-0.75*y.^2-3.75*x.^2-1.5*x).*(x+y>1)+...
      0.7575*exp(-y.^2-6*x.^2).*((x+y>-1) & (x+y<=1))+...
      0.5457*exp(-0.75*y.^2-3.75*x.^2+1.5*x).*(x+y<=-1);
   surf(x,y,z), xlim([-1.5 1.5]); shading flat
```

and the three-dimensional surface can be shown in Figure 2.16.

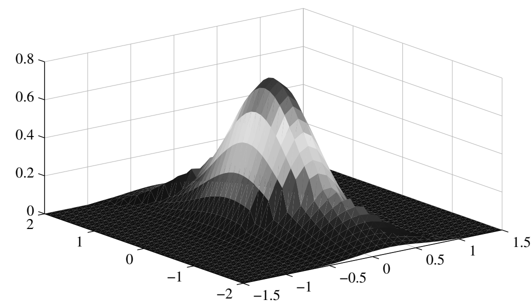
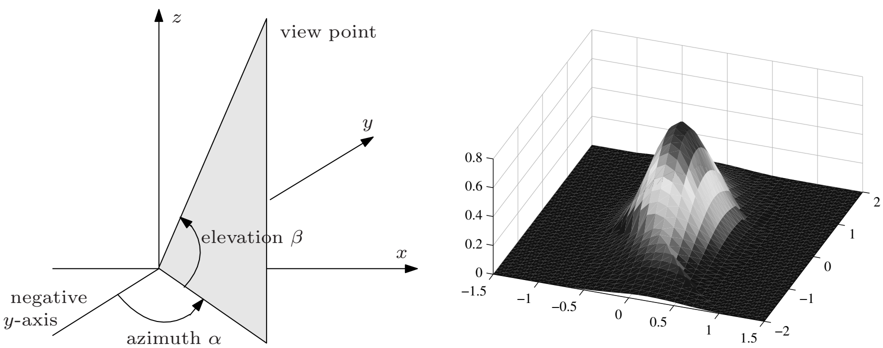


FIGURE 2.16: Surface of a piecewise function with two variables

2.6.3 Viewpoint setting in 3D graphs

In the MATLAB 3D graphics facilities, viewpoint setting functions are provided, which allows the user to view the plot from any angle. Two ways are provided: one is the toolbar facility in the figure window, and the other is the `view()` function.

An illustration to the definition of the viewpoint is given in Figure 2.17 (a), where the two angles α and β can be used to define uniquely the viewpoint. The azimuth α is defined as the angle between the projection line in x-y plane with the negative y-axis, with a default value of $\alpha = -37.5°$. The elevation β is defined as the angle with the x-y plane, with a default value of $\beta = 30°$.



(a) definition of viewpoints (b) 3D surface after viewpoint change

FIGURE 2.17: Viewpoint settings of three-dimensional surfaces

The function `view(`α`,`β`)` can be used to set the viewpoint, where the angles α and β are the azimuth and elevation angles respectively. For instance, the setting `view(0,90)` shows the planform, while `view(0,0)` and `view(90,0)` show the front view and the side elevation respectively.

For instance, one may change the viewpoint in the three-dimensional surface display shown in Figure 2.16. One may set $\alpha = 20°$, and $\beta = 50°$, the following statements can be used and the results shown in Figure 2.17 (b) can be obtained.

```
>> view(20,50), xlim([-1.5 1.5]) % set the range of x-axis
```

Example 2.32 Consider again the surface plot in Example 2.29. View the surface from different angles.

Solution The surface plots from different viewpoints can be obtained using the following statements, as shown in Figure 2.18.

```
>> [x,y] = meshgrid(-3:0.1:3,-2:0.1:2);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
   subplot(221), surf(x,y,z), view(0,90); % planform
```

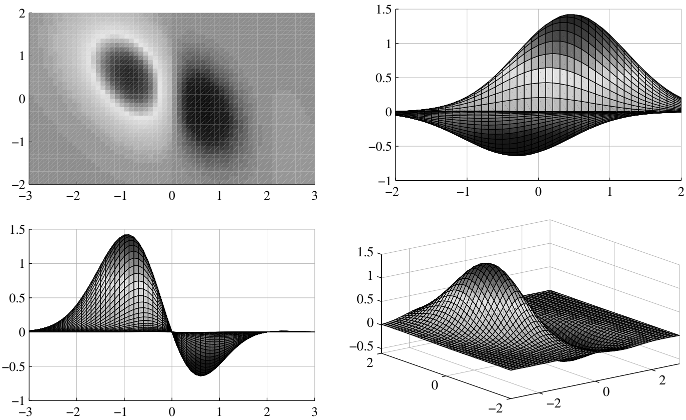


FIGURE 2.18: Surface view from different angles

```
subplot(222), surf(x,y,z), view(90,0); % side elevation
subplot(223), surf(x,y,z), view(0,0);  % front view
subplot(224), surf(x,y,z),             % 3D surface plot
```

Exercises

1. In MATLAB environment, the following statements can be given
   `tic, A=rand(500); B=inv(A); norm(A*B-eye(500)), toc`
   Run the statements and observe results. If you are not sure with the commands, just use the on-line `help` facilities to display information on the related functions. Then explain in detail the statement and the results.
2. Suppose that a polynomial can be expressed by $f(x) = x^5 + 3x^4 + 4x^3 + 2x^2 + 3x + 6$. If one wants to substitute x by $\dfrac{s-1}{s+1}$, the function $f(x)$ can be changed into a function of s. Use the Symbolic Math Toolbox to do the substitution and get the simplest result.
3. Input the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ into MATLAB workspace where
$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 2 & 3 & 4 & 1 \\ 3 & 2 & 4 & 1 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1+\mathrm{j}4 & 2+\mathrm{j}3 & 3+\mathrm{j}2 & 4+\mathrm{j}1 \\ 4+\mathrm{j}1 & 3+\mathrm{j}2 & 2+\mathrm{j}3 & 1+\mathrm{j}4 \\ 2+\mathrm{j}3 & 3+\mathrm{j}2 & 4+\mathrm{j}1 & 1+\mathrm{j}4 \\ 3+\mathrm{j}2 & 2+\mathrm{j}3 & 4+\mathrm{j}1 & 1+\mathrm{j}4 \end{bmatrix}.$$
   It is seen that $\boldsymbol{A}$ is a 4×4 matrix. If a command $\boldsymbol{A}(5, 6) = 5$ is given, what will happen?
4. For a matrix $\boldsymbol{A}$, if one wants to extract all the even rows to form matrix $\boldsymbol{B}$, what command should be used? Suppose that matrix $\boldsymbol{A}$ is defined by $\boldsymbol{A}$ =`magic(8)`, establish matrix $\boldsymbol{B}$ with suitable statements and see whether the results are correct.

5. Implement the following piecewise function where $\boldsymbol{x}$ can be given by scalar, vectors, matrices or even other multi-dimensional arrays, the returned argument $\boldsymbol{y}$ should be the same size as that of $\boldsymbol{x}$. The parameters h and D are scalars.
$$\boldsymbol{y}=f(\boldsymbol{x})=\begin{cases} h, & \boldsymbol{x}>D \\ h/D\boldsymbol{x}, & |\,\boldsymbol{x}\,|\leqslant D \\ -h, & \boldsymbol{x}<-D \end{cases}$$
6. Evaluate using numerical method the sum $S=1+2+4+\cdots+2^{62}+2^{63}=\sum_{i=0}^{63}2^i$, the use of vectorized form is suggested. Check whether accurate solutions can be found and why. Find the accurate sum using the symbolic computation methods.
7. Write an M-function `mat_add()` with the syntax
   $\boldsymbol{A}$`=mat_add(`$\boldsymbol{A}_1$`,`$\boldsymbol{A}_2$`,`$\boldsymbol{A}_3$`,`$\cdots$`)`
   In the function, it is required that arbitrary number of input arguments $\boldsymbol{A}_i$ are allowed.
8. A MATLAB function can be written whose syntax is
   $\boldsymbol{v}$`=[`$h_1, h_2, h_m, h_{m+1}, \cdots, h_{2m-1}$`]` and $\boldsymbol{H}$`=myhankel(`$\boldsymbol{v}$`)`
   where the vector $\boldsymbol{v}$ is defined, and out of it, the output argument should be an $m\times m$ Hankel matrix.
9. From matrix theory, it is known that if a matrix $\boldsymbol{M}$ is expressed as $\boldsymbol{M}=\boldsymbol{A}+\boldsymbol{BCB}^{\mathrm{T}}$, where $\boldsymbol{A}$, $\boldsymbol{B}$ and $\boldsymbol{C}$ are the matrices of relevant sizes, the inverse of $\boldsymbol{M}$ can be calculated by the following algorithm
$$\boldsymbol{M}^{-1}=\left(\boldsymbol{A}+\boldsymbol{BCB}^{\mathrm{T}}\right)^{-1}=\boldsymbol{A}^{-1}-\boldsymbol{A}^{-1}\boldsymbol{B}\left(\boldsymbol{C}^{-1}+\boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{B}\right)^{-1}\boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}^{-1}$$
   The matrix inversion can be carried out using the formula easily. Suppose that there is a 5×5 matrix $\boldsymbol{M}$, from which the three other matrices can be found.
$$\boldsymbol{M}=\begin{bmatrix} -1 & -1 & -1 & 1 & 0 \\ -2 & 0 & 0 & -1 & 0 \\ -6 & -4 & -1 & -1 & -2 \\ -1 & -1 & 0 & 2 & 0 \\ -4 & -3 & -3 & -1 & 3 \end{bmatrix},\quad \boldsymbol{A}=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$
$$\boldsymbol{B}=\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 2 & 1 & 0 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix},\quad \boldsymbol{C}=\begin{bmatrix} 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & -1 & 0 & 1 \end{bmatrix}.$$
   Write the statement to evaluate the inverse matrix. Check the accuracy of the inversion. Compare the accuracy of the inversion method and the direct inversion method with `inv()` function.
10. Consider the following iterative model
$$\begin{cases} x_{k+1}=1+y_k-1.4x_k^2 \\ y_{k+1}=0.3x_k \end{cases}$$
   with initial conditions $x_0=0$, $y_0=0$. Write an M-function to evaluate the sequence x_i, y_i. 30000 points can be obtained by the function to construct the $\boldsymbol{x}$ and $\boldsymbol{y}$ vectors. The points can be expressed by a dot, rather than lines. In

this case, the so-called Hénon attractor can be drawn.

11. A regular triangle can be drawn by MATLAB statements easily. Use the loop structure to design an M-function that, in the same coordinates, a sequence of regular triangles can be drawn, each by rotating a small angle from the previous one.
12. Select suitable step-sizes and draw the function curve for $\sin(1/t)$, where $t \in (-1, 1)$.
13. For suitably assigned ranges of θ, draw polar plots for the following functions.
(i) $\rho = 1.0013\theta^2$, (ii) $\rho = \cos(7\theta/2)$,
(iii) $\rho = \sin(\theta)/\theta$, (iv) $\rho = 1 - \cos^3(7\theta)$
14. Find the solutions to the following equations using graphical methods and verify the solutions.
$$\begin{cases} x^2 + y^2 = 3xy^2 \\ x^3 - x^2 = y^2 - y \end{cases}$$
15. Draw the 3D surface plots for the functions xy and $\sin(xy)$ respectively. Also draw the contours of the functions. View the 3D surface plot from different angles.
16. In graphics command, there is a trick in hiding certain parts of the plot. If the function values are assigned to `NaN`s, the point on the curve or the surface will not be shown. Draw first the surface plot of the function $z = \sin xy$. Then cut off the region that satisfies $x^2 + y^2 \leqslant 0.5^2$.

Chapter 3

Calculus Problems

The calculus established by Isaac Newton and Gottfried Wilhelm Leibniz is fundamental to many branches of sciences and engineering. In traditional calculus courses, limits, differentiations, integrals, series expansions such as Taylor series and Fourier series expansions for single-variable and multivariable functions are the main topics discussed. The analytical solutions to these problems can be obtained by the direct use of the corresponding functions provided by the Symbolic Math Toolbox of MATLAB which will be discussed in Section 3.1. The Taylor series expansions for single- and multivariable functions as well as the Fourier series expansions are discussed in Section 3.2. Moreover, the series summation and product problems are discussed. Sections 3.5 and 3.6 present methods for path integrals, line integrals and surface integrals. Most of the materials presented in this chapter are symbolic-based, which cannot be solved using conventional computer programming languages such as C for average users. Computer mathematics languages such as MATLAB should be used instead.

In many scientific and engineering researches, the analytical solutions to calculus problems may face difficulties, since the original functions may not be given explicitly. For problems with measured data, numerical differentiations and integrals should be applied accordingly. They are illustrated in Sections 3.3 and 3.4, respectively. Alternative solutions to the same numerical calculus problems using spline interpolation are given in Chapter 8.

As an extension to the traditional (integer-order) calculus, non-integer-order or fractional-order calculus, will be discussed in Chapter 10.

For readers who wish to check the detailed explanations of calculus, we recommend the free textbooks [7, 8].

3.1 Analytical Solutions to Calculus Problems

The Symbolic Math Toolbox of MATLAB can be used directly in solving the limit problems, the differentiation problems, and the integral problems. Using the methods presented in this section, the readers will be equipped with the ability in solving ordinary calculus problems directly by computers.

3.1.1 Analytical solutions to limit problems

Limits of single-variable functions

Assume that the function to be analyzed is $f(x)$, the limit is defined as

$$L = \lim_{x \to x_0} f(x) \tag{3.1}$$

where x_0 can be either a given value or infinity. For certain functions, the left or right limit can be defined as

$$L_1 = \lim_{x \to x_0^-} f(x), \quad \text{or} \quad L_2 = \lim_{x \to x_0^+} f(x) \tag{3.2}$$

where the former means to approach the point x_0 from the left-hand side which is referred to as the *left limit* problem. The latter is referred to as the *right limit* problem. The limit problems summarized above can be solved by the use of the `limit()` function, where

L=`limit(`*fun*`,`x`,`x_0`)`	% calculate the limit
L=`limit(`*fun*`,`x`,`x_0`,'left'` or `'right')`	% the one-sided limit

To use the functions in Symbolic Math Toolbox, symbolic variables such as x should be declared first. Then, the limit function *fun* can be expressed. If x_0 is ∞, one can assign it to `inf`. If the one-sided limit is required, the `'left'` or `'right'` option should be specified. The following examples are used to demonstrate the use of the `limit()` function in MATLAB.

Example 3.1 Solve the limit problem $\lim\limits_{x \to \infty} x \left(1 + \dfrac{a}{x}\right)^x \sin \dfrac{b}{x}$.

Solution For this problem, one should first declare the variables a, b and x as symbolic variables. Then the function can be defined and the `limit()` function can be called directly to solve the problem, which returns $L = \mathrm{e}^a b$.

```
>> syms x a b; f=x*(1+a/x)^x*sin(b/x); L=limit(f,x,inf)
```

Example 3.2 Solve the one-sided limit problem $\lim\limits_{x \to 0^+} \dfrac{\mathrm{e}^{x^3} - 1}{1 - \cos\sqrt{x - \sin x}}$.

Solution With the `limit()` function, the one-sided limit can easily be solved, with the limit of 12.

```
>> syms x; limit((exp(x^3)-1)/(1-cos(sqrt(x-sin(x)))),x,0,'right')
```

One can further verify the above problem graphically over a proper range of interest. For instance, if the interval $(-0.1, 0.1)$ is considered, the function over the interval can be drawn in Figure 3.1.

```
>> x=-0.1:0.001:0.1; y=(exp(x.^3)-1)./(1-cos(sqrt(x-sin(x))));
   plot(x,y,'-',[0],[12],'o')
```

It can be seen that the limit of the original problem is also 12.

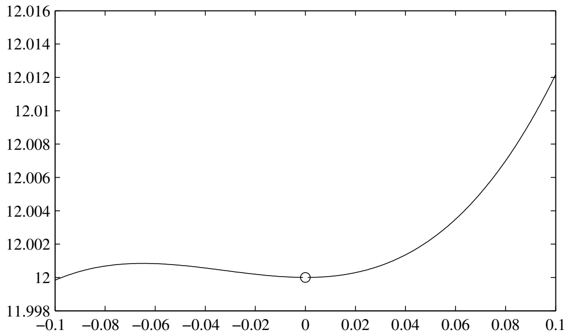


FIGURE 3.1: The curve of the function around $x = 0$

```
>> syms x; limit((exp(x^3)-1)/(1-cos(sqrt(x-sin(x)))),x,0)
```

Consider again the original problem. The aim of the original one-sided limit requirement ensures that the expression under the square root is positive. In fact, for imaginary variables, one can still find from the Euler's formula that $\cos \mathrm{j}\alpha = (\mathrm{e}^{\alpha} + \mathrm{e}^{-\alpha})/2$. Thus the one-sided limits for the function are the same for this example, which further verifies that the original function is continuous around $x = 0$ as also seen from Figure 3.1.

Limits of multivariable functions

The limit problems for multivariable functions can also be solved with the MATLAB function `limit()`. For instance, the limit to the function $f(x, y)$

$$L = \lim_{\substack{x \to x_0 \\ y \to y_0}} f(x, y) \tag{3.3}$$

can be solved by the nested use of the `limit()` function. For example, L_1=`limit(limit(`*fun*,x,x_0`)`,y,y_0`)` or L_1=`limit(limit(`*fun*,y,y_0`)`,x,x_0`)` where x_0 and y_0 can be either constants or functions of another variable, for instance $x \to g(y)$. In the latter case, the order of the function call cannot be changed.

Example 3.3 Solve the limit problem $\displaystyle \lim_{\substack{x \to 1/\sqrt{y} \\ y \to \infty}} \mathrm{e}^{-1/(y^2+x^2)} \frac{\sin^2 x}{x^2} \left(1 + \frac{1}{y^2}\right)^{x+a^2y^2}$.

Solution The problem can easily be solved with the following MATLAB scripts

```
>> syms x y a; f=exp(-1/(y^2+x^2))*sin(x)^2/x^2*(1+1/y^2)^(x+a^2*y^2);
   L=limit(limit(f,x,1/sqrt(y)),y,inf)
```

which yields $L = \mathrm{e}^{a^2}$.

3.1.2 Analytical solutions to derivative problems

Derivative and high-order derivatives

If the function is known, the function `diff()` can be used to calculate its derivatives. The syntaxes of the `diff()` function are

```
y=diff(fun,x)      % find the derivative
y=diff(fun,x,n)    % evaluate the nth order derivative
```

where *fun* is the symbolic expression of a given function; x is the symbolic independent variable; n is the order of the derivative to be taken.

Example 3.4 Compute $\dfrac{\mathrm{d}^4 f(x)}{\mathrm{d}x^4}$ for a given function $f(x) = \dfrac{\sin x}{x^2+4x+3}$.

Solution It should be noted that this is the first example given at the beginning of the book. The derivatives can easily be obtained with the following MATLAB functions. The variable x should be declared as a symbolic variable first, then the function `diff()` can be called to find the first-order derivative.

```
>> syms x; f=sin(x)/(x^2+4*x+3); f1=diff(f)
```

The readability of the results directly obtained may not be very high. It is suggested that the results should be converted with the use of `pretty()` function, or by `latex()` function. The latter can be used to convert the result into the form in the well-known LaTeX string, the best scientific documentation system. Under LaTeX, the result can be better displayed as $\dfrac{\cos x}{x^2+4x+3} - \dfrac{\sin x\,(2x+4)}{(x^2+4x+3)^2}$. It can be seen that the quality of LaTeX display is far better than the one obtained in MATLAB. In the later description, the LaTeX display will be extensively used to increase the readability.

The original function and the first-order derivative function can easily be obtained and their respective curves are shown in Figure 3.2.

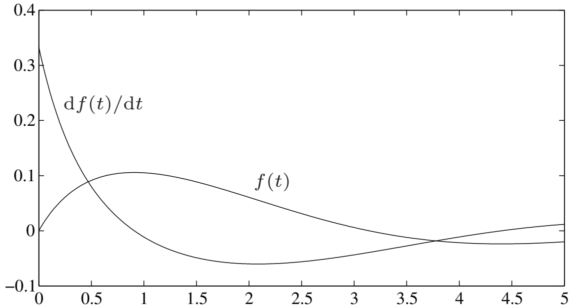


FIGURE 3.2: The curves of the original function and its derivative

```
>> x1=0:.01:5; y=subs(f,x,x1); y1=subs(f1,x,x1); plot(x1,y,x1,y1,':')
```

The fourth-order derivative can be simply calculated from

```
>> f4=diff(f,x,4)
```

and the result is displayed in LaTeX

$$\frac{\sin x}{x^2+4x+3}+4\frac{(2x+4)\cos x}{(x^2+4x+3)^2}-12\frac{(2x+4)^2\sin x}{(x^2+4x+3)^3}+12\frac{\sin x}{(x^2+4x+3)^2}-24\frac{(2x+4)^3\cos x}{(x^2+4x+3)^4}$$
$$+48\frac{(2x+4)\cos x}{(x^2+4x+3)^3}+24\frac{(2x+4)^4\sin x}{(x^2+4x+3)^5}-72\frac{(2x+4)^2\sin x}{(x^2+4x+3)^4}+24\frac{\sin x}{(x^2+4x+3)^3}.$$

From the above simplified results, it is clear that the direct use of the function `simple()` is not sufficient for this example. For the given example, it can immediately be found that one may extract the terms $\sin x$ and $\cos x$ from the results and the coefficients for these terms can be simplified separately such that

```
>> collect(simple(f4),sin(x)), collect(simple(f4),cos(x))
```

The even more concise results can be obtained shown as follows:

$$\frac{\mathrm{d}^4 f(x)}{\mathrm{d}x^4}=8(x^5+10x^4+26x^3-4x^2-99x-102)\frac{\cos x}{(x^2+4x+3)^4}+$$
$$(x^8+16x^7+72x^6-32x^5-1094x^4-3120x^3-3120x^2+192x+1581)\frac{\sin x}{(x^2+4x+3)^5}.$$

The differentiation function `diff()` can easily be used to find high-order derivatives. For instance, the 100th order derivative of the same function can be found within one second.

```
>> tic, diff(f,x,100); toc
```

Partial derivatives of multivariable functions

There is no direct function which can be used in finding the partial derivatives in MATLAB. The function `diff()` can actually be used instead. For instance, if a function $f(x,y)$ with two variables is defined, the partial derivative $\partial^{m+n}f/(\partial x^m \partial y^n)$ can be evaluated by the nested use of the `diff()` function as follows:

`f=diff(diff(fun,x,m),y,n)` , or `f=diff(diff(fun,y,n),x,m)`

Example 3.5 Find the partial derivatives of $z=f(x,y)=(x^2-2x)\mathrm{e}^{-x^2-y^2-xy}$ function and investigate the function further using graphical method.

Solution The partial derivatives $\partial z/\partial x$ and $\partial z/\partial y$ can be evaluated easily using

```
>> syms x y; z=(x^2-2*x)*exp(-x^2-y^2-x*y);
   zx=simple(diff(z,x)), zy=diff(z,y)
```

and the mathematical representations of the derivatives are

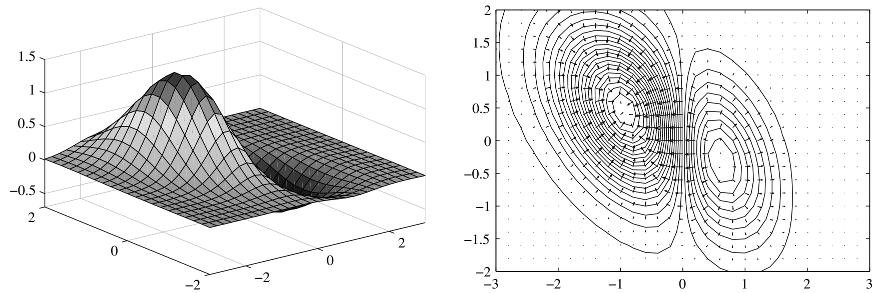
$$\frac{\partial z(x,y)}{\partial x}=-\mathrm{e}^{-x^2-y^2-xy}(-2x+2+2x^3+x^2y-4x^2-2xy)$$
$$\frac{\partial z(x,y)}{\partial y}=-x(x-2)(2y+x)\mathrm{e}^{-x^2-y^2-xy}.$$

Within the rectangular region where $x\in(-3,3), y\in(-2,2)$, mesh grids can be defined and the partial derivatives can be obtained numerically over the mesh grids. The three-dimensional surface of the original function is shown in Figure 3.3 (a)

```
>> [x0,y0]=meshgrid(-3:.2:3,-2:.2:2);
   z0=subs(z,{x,y},{x0,y0}); % substituting the two variables
   surf(x0,y0,z0), axis([-3 3 -2 2 -0.7 1.5]) % three-dimensional surface
```

From the partial derivatives obtained, the numerical solutions at the mesh grids can be evaluated. The function `quiver()` can then be used to draw attractive curves, and the curves can be superimposed over the contour of the original function with the following statements, as shown in Figure 3.3 (b).

```
>> contour(x0,y0,z0,30), hold on    % contours of the function
   zx0=subs(zx,{x,y},{x0,y0}); zy0=subs(zy,{x,y},{x0,y0});
   quiver(x0,y0,zx0,zy0)            % draw the attractive curves
```



(a) three-dimensional surface (b) contours with gradients

FIGURE 3.3: Graphical interpretation of the functions with two variables

Example 3.6 For a given function with three independent variables x, y and z such that $f(x,y,z) = \sin(x^2y)\mathrm{e}^{-x^2y-z^2}$, find the partial derivative $\partial^4 f(x,y,z)/(\partial x^2 \partial y \partial z)$.

Solution The following MATLAB statements can be given to solve this problem

```
>> syms x y z; f=sin(x^2*y)*exp(-x^2*y-z^2);
   df=diff(diff(diff(f,x,2),y),z); df=simple(df)
```

the results can be obtained as

$$-4z\mathrm{e}^{-x^2y-z^2}\left[\cos x^2y - 10yx^2\cos x^2y + 4x^4y^2\sin x^2y + 4x^4y^2\cos x^2y - \sin x^2y\right].$$

Jacobian matrix of multivariable functions

Assume that there are n independent variables, and m functions defined as

$$\begin{cases} y_1 = f_1(x_1, x_2, \cdots, x_n) \\ y_2 = f_2(x_1, x_2, \cdots, x_n) \\ \quad\vdots \qquad \vdots \\ y_m = f_m(x_1, x_2, \cdots, x_n). \end{cases} \tag{3.4}$$

The partial derivative $\partial y_i/\partial x_j$ for each combination of i and j can be represented in the matrix form as

$$\boldsymbol{J} = \begin{bmatrix} \partial y_1/\partial x_1 & \partial y_1/\partial x_2 & \cdots & \partial y_1/\partial x_n \\ \partial y_2/\partial x_1 & \partial y_2/\partial x_2 & \cdots & \partial y_2/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial y_m/\partial x_1 & \partial y_m/\partial x_2 & \cdots & \partial y_m/\partial x_n \end{bmatrix} \tag{3.5}$$

and such a matrix is referred to as the *Jacobian matrix*. Jacobian matrices are quite useful in many research areas, such as robotics and image processing. Jacobian matrix can be obtained using the `jacobian()` function of the Symbolic Math Toolbox directly. The syntax of the function is `J=jacobian(y,x)`, where $\boldsymbol{x}$ is the vector of independent variables, and $\boldsymbol{y}$ is the vector of multivariable functions.

Example 3.7 Consider that the functions for coordinate transformation are defined as $x = r\sin\theta\cos\phi$, $y = r\sin\theta\sin\phi$, and $z = r\cos\theta$. Find the Jacobian matrix of these functions.

Solution Three independent variables can be declared and the three functions can then be expressed. The following statements can be used to find the Jacobian matrix

```
>> syms r theta phi; x=r*sin(theta)*cos(phi);
   y=r*sin(theta)*sin(phi); z=r*cos(theta);
   J=jacobian([x; y; z],[r theta phi])
```

The Jacobian matrix is obtained as

$$\boldsymbol{J} = \begin{bmatrix} \sin\theta\cos\phi & r\cos\theta\cos\phi & -r\sin\theta\sin\phi \\ \sin\theta\sin\phi & r\cos\theta\sin\phi & r\sin\theta\cos\phi \\ \cos\theta & -r\sin\theta & 0 \end{bmatrix}.$$

Partial derivatives of implicit functions

Assume that an implicit function is defined as $f(x_1, x_2, \cdots, x_n) = 0$. The partial derivative $\partial x_i/\partial x_j$ among the independent variables can be obtained using the following formula

$$\frac{\partial x_i}{\partial x_j} = -\frac{\dfrac{\partial}{\partial x_j} f(x_1, x_2, \cdots, x_n)}{\dfrac{\partial}{\partial x_i} f(x_1, x_2, \cdots, x_n)}. \tag{3.6}$$

Since the derivatives of f with respect to x_i and x_j can easily be obtained separately with the function `diff()`, the partial derivative of $\partial x_i/\partial x_j$ can be obtained directly using the MATLAB functions `F=-diff(f,x_j)/diff(f,x_i)`.

Example 3.8 Consider again the implicit function $f(x, y) = (x^2-2x)\mathrm{e}^{-x^2-y^2-xy} = 0$. Evaluate $\partial y/\partial x$.

Solution It can be found from (3.6) that the partial derivative $\partial y/\partial x$ can be obtained with the following statements

```
>> syms x y; f=(x^2-2*x)*exp(-x^2-y^2-x*y);
   simple(-diff(f,x)/diff(f,y))
```

The result is $\dfrac{2x-2-2x^3-x^2y+4x^2+2xy}{x\,(x-2)\,(2y+x)}$.

Derivatives of parametric equations

When the function $y(x)$ is given as parametric equations $y=f(t), x=g(t)$, the kth order derivative of the function $\dfrac{\mathrm{d}^k y}{\mathrm{d}x^k}$ can be calculated using the following formula

$$
\begin{aligned}
&\frac{\mathrm{d}y}{\mathrm{d}x}=\frac{f'(t)}{g'(t)}\\
&\frac{\mathrm{d}^2y}{\mathrm{d}x^2}=\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{f'(t)}{g'(t)}\right)\frac{1}{g'(t)}=\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}y}{\mathrm{d}x}\right)\frac{1}{g'(t)}\\
&\vdots\\
&\frac{\mathrm{d}^ny}{\mathrm{d}x^n}=\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}^{n-1}y}{\mathrm{d}x^{n-1}}\right)\frac{1}{g'(t)}.
\end{aligned}
\tag{3.7}
$$

Using the recursive calling structure, the following MATLAB function can be written to implement the above algorithm and the function should be placed in the `@sym` directory

```
function result=paradiff(y,x,t,n)
if mod(n,1)~=0, error('n should positive integer, please correct')
else
   if  n==1, result=diff(y,t)/diff(x,t);
   else, result=diff(paradiff(y,x,t,n-1),t)/diff(x,t);
end, end
```

Example 3.9 For the parametric equations $y=\dfrac{\sin t}{(t+1)^3}$, $x=\dfrac{\cos t}{(t+1)^3}$, find $\dfrac{\mathrm{d}^3y}{\mathrm{d}x^3}$.

Solution From the above parametric equations, the derivative can be found by

```
>> syms t; y=sin(t)/(t+1)^3; x=cos(t)/(t+1)^3;
   f=paradiff(y,x,t,3); [n,d]=numden(f); F=simple(n)/simple(d)
```

The results can be simplified into the following form:

$$\frac{\mathrm{d}^3y}{\mathrm{d}x^3}=\frac{-3(t+1)^7\left[(t^4+4t^3+6t^2+4t-23)\cos t-(4t^3+12t^2+32t+24)\sin t\right]}{(t\sin t+\sin t+3\cos t)^5}.$$

3.1.3 Analytical solutions to integral problems

In calculus, integral problems are often described mathematically as

$$\int f(x)\,\mathrm{d}x,\ \int_a^b f(x)\,\mathrm{d}x,\ \int\cdots\int f(x_1,x_2,\cdots,x_n)\,\mathrm{d}x_n\cdots\mathrm{d}x_2\mathrm{d}x_1 \tag{3.8}$$

where function $f(\cdot)$ is referred to as the *integrand*. The first integral is referred to as the *indefinite integral*, while $F(x)$ is referred to as the *primitive function*. The other two integrals are respectively referred to as the *definite integral* and *multiple integrals*. To solve the integral problems, according to calculus courses, one has to select, largely by experience, the integration methods, such as integration by substitution, or integration by parts, or others. Thus, solving integral problems could be a tedious task.

Indefinite integrals

The `int()` function provided in the Symbolic Math Toolbox of MATLAB can be used to evaluate the indefinite integrals to given functions. The syntax of the function is `F=int(fun,x)`, where the integrand can be described by *fun*. If only one variable appears in the integrand, the argument x can be omitted. The returned argument is the primitive $F(x)$. In fact, the general solution to the indefinite integral problem is $F(x)+C$, with C an arbitrary constant.

For any integrable functions, the use of the function `int()` can reduce the complicated work such that the primitive function can be obtained directly. However, for symbolically non-integrable functions, the `int()` function may not give useful results. In this case, numerical methods have to be used instead.

Example 3.10 Consider the function given in Example 3.4. The `diff()` function can be used to find the derivative of $f(x)$. If the indefinite integral is made upon the results, check whether the original function can be restored.

Solution The original function can be defined and the integral can be taken on the first-order derivative such that

```
>> syms x; y=sin(x)/(x^2+4*x+3); y1=diff(y); y0=int(y1)
```

the result is then $\dfrac{\sin x}{2(x+1)}-\dfrac{\sin x}{2(x+3)}$. It can be seen that the result restores the original function.

Now consider taking the fourth-order derivative to the original function by applying `int()` four times in a nested way as follows:

```
>> y4=diff(y,4); y0=int(int(int(int(y4)))); simple(y0)
```

and the result is $\dfrac{\sin x}{(x+1)(x+3)}$, which is still the same as the original function.

Example 3.11 Show that

$$\int x^3\cos^2 ax\,\mathrm{d}x=\frac{x^4}{8}+\left(\frac{x^3}{4a}-\frac{3x}{8a^3}\right)\sin 2ax+\left(\frac{3x^2}{8a^2}-\frac{3}{16a^4}\right)\cos 2ax+C.$$

Solution The following MATLAB statements can be used:

```
>> syms a x; f=simple(int(x^3*cos(a*x)^2,x))
```

and the simplified results can be obtained as

$$\frac{1}{16a^4}\Big[4a^3x^3\sin(2ax)+2a^4x^4+6a^2x^2\cos(2ax)-6\,ax\sin(2ax)+3-3\cos(2ax)\Big].$$

It can be seen that the result is not the same as the one on the right-hand side. Let us check the difference. Using the following scripts

```
>> f1=x^4/8+(x^3/(4*a)-3*x/(8*a^3))*sin(2*a*x)+...
      (3*x^2/(8*a^2)-3/(16*a^4))*cos(2*a*x);
   simple(f-f1)  % difference is taken and simplified
```

after simplification, the difference is $-3/(16a^4)$ which is not zero. However, fortunately, since the difference between the two primitive functions is a constant, it can be included into the final constant C. Thus the original equation can be proved.

Example 3.12 Consider the two integrands

$$f(x)=\mathrm{e}^{-x^2/2},\quad \text{and}\quad g(x)=x\sin(ax^4)\mathrm{e}^{x^2/2}.$$

They are all known to be not integrable. Compute the integral to the two functions.

Solution Let us consider the integral to the first integrand $f(x)=\mathrm{e}^{-x^2/2}$. The following MATLAB functions can be used

```
>> syms x; int(exp(-x^2/2))
```

and the result obtained is $\mathrm{erf}(\sqrt{2})/\sqrt{2\pi}$. Since the original integrand is not integrable, a function $\mathrm{erf}(x)=\dfrac{2}{\sqrt{\pi}}\displaystyle\int_0^x \mathrm{e}^{-t^2}\mathrm{d}t$ is introduced. Thus the "analytical" solution to the original problem can be obtained.

The second integrand can be tested under the `int()` function. The following MATLAB statements

```
>> syms a x; int(x*sin(a*x^4)*exp(x^2/2))
```

result in the following returned warning message, which means that the explicit solutions cannot be obtained.

```
Warning: Explicit integral could not be found.
> In sym.int at 58
ans =
   int(x*sin(a*x^4)*exp(1/2*x^2),x)
```

Computing definite and infinite integrals

The definite integrals and infinite integrals are also part of calculus. For instance, although the function $\mathrm{erf}(x)$ is defined previously, the integral of a particular value of x cannot be obtained analytically. In this case, definite integrals, in cooperation with numerical methods, can be obtained. The function `int()` can be used to evaluate the definite and infinite integrals. The syntax of the function is `I=int(fun,x,a,b)`, where x is the independent variable, (a, b) is the integral interval. For infinite integrals, the arguments a and b can be assigned to `-Inf` or `Inf`. Also if no exact value can be obtained directly, the `vpa()` function can be used to evaluate the solutions numerically.

Example 3.13 Consider the integrands given previously in Example 3.12. When $a = 0$, $b = 1.5$ (or ∞), evaluate the values of the integral.

Solution The following statements can be used in solving the definite and infinite integral problems

```
>> syms x; I1=int(exp(-x^2/2),x,0,1.5)
   vpa(I1,70), I2=int(exp(-x^2/2),x,0,inf)
```

where $I_1 = \sqrt{\pi/2}\,\mathrm{erf}[3/(2\sqrt{2})]$, and the high-precision numerical solution to the definite integral is $I_1 = 1.085853317666016569702419076542265042534236293532156326 72991722930853$. The analytical solution to the infinite integral is $I_2 = \sqrt{\pi/2}$.

Example 3.14 Solve the definite integral problems for functional boundaries

$$I(t) = \int_{\cos t}^{\mathrm{e}^{-2t}} \frac{-2x^2+1}{(2x^2-3x+1)^2}\,\mathrm{d}x.$$

Solution The function `int()` can be used in solving definite integrals and the following statements can be used

```
>> syms x t; f=(-2*x^2+1)/(2*x^2-3*x+1)^2;
   I=simple(int(f,x,cos(t),exp(-2*t))),
```

and the results can be expressed as

$$I(t) = -\frac{(2\mathrm{e}^{-2t}\cos t - 1)(\mathrm{e}^{-2t} - \cos t)}{(\mathrm{e}^{-2t} - 1)(2\mathrm{e}^{-2t} - 1)(\cos t - 1)(2\cos t - 1)}.$$

Computing multiple integrals

Multiple integral problems can also be solved by using the same MATLAB function `int()`. Generally speaking, usually the inner integrals should be carried out first and then outer integrals. However, the sequence of integrals should be observed. In each integration step, the `int()` function can be used. Thus sometimes in certain integration steps, the inner integral may not yield a primitive function, which results in no analytical solution to the overall integral problem. If the sequence of integrals can be changed, analytical solutions may be obtained. Numerical solutions to multiple integral problems will be presented in Section 3.4.3.

Example 3.15 Compute the multiple integrals $\int \cdots \int F(x, y, z)\,\mathrm{d}x^2\mathrm{d}y\mathrm{d}z$ where the integrand $F(x, y, z)$ is defined as

$$-4z\mathrm{e}^{-x^2y-z^2}\left[\cos x^2y - 10yx^2\cos x^2y + 4x^4y^2\sin x^2y + 4x^4y^2\cos x^2y - \sin x^2y\right].$$

Solution In fact, the above $F(x, y, z)$ function was obtained by taking partial derivatives to the function $f(x, y, z)$ defined in Example 3.6. Thus taking inverse operations in this example should restore the same primitive function.

One may integrate once with respect to z, once to y and twice to x. The following results can be obtained through simplification

```
>> syms x y z;
   f0=-4*z*exp(-x^2*y-z^2)*(cos(x^2*y)-10*cos(x^2*y)*y*x^2+...
      4*sin(x^2*y)*x^4*y^2+4*cos(x^2*y)*x^4*y^2-sin(x^2*y));
   f1=int(f0,z); f1=int(f1,y); f1=int(f1,x); f1=simple(int(f1,x))
```

with the primitive function $\sin(x^2y)\mathrm{e}^{-x^2y-z^2}$, which is exactly the same as the function defined in Example 3.6. Now if one alters the order of integration, i.e., change the order to $z \to x \to x \to y$

```
>> f2=int(f0,z); f2=int(f2,x); f2=int(f2,x); f2=simple(int(f2,y))
```

the result becomes $2\dfrac{\mathrm{e}^{-x^2y-z^2}\tan\left(x^2y/2\right)}{1+\tan^2\left(x^2y/2\right)}$. The primitive function obtained does not look the same as the original function in Example 3.6. If one simplifies the difference between the two functions, i.e., `simple(f1-f2)`, it can be seen that the difference is 0, which means that the two functions are identical.

Example 3.16 Compute the definite integral $\displaystyle\int_0^2\int_0^\pi\int_0^\pi 4xz\mathrm{e}^{-x^2y-z^2}\mathrm{d}z\mathrm{d}y\mathrm{d}x$.

Solution The following statements can be given to calculate the triple definite integral

```
>> syms x y z
   int(int(int(4*x*z*exp(-x^2*y-z^2),x,0,2),y,0,pi),z,0,pi)
```

and the results obtained are

```
pi*Ei(1,4*pi)*(1/pi-1/pi*exp(-pi^2))+pi*log(pi)*(1/pi-1/pi*exp(-pi^2))+
pi*eulergamma*(1/pi-1/pi*exp(-pi^2))+2*pi*log(2)*(1/pi-1/pi*exp(-pi^2))
```

where `eulergamma` is the Euler constant γ, $\mathrm{Ei}(n,z)=\displaystyle\int_1^\infty \mathrm{e}^{-zt}t^{-n}\,\mathrm{d}t$ is an exponential integral. The integrand is not integrable analytically. However, numerical solutions can be found. Thus the accurate numerical solution to the original problem can be found from `vpa(ans)` command and the integral value is 3.10807940208541272.

3.2 Series Expansions and Series Evaluations

Taylor series expansions to functions with a single variable and multiple variables will be discussed in this section. The Fourier series expansion to given functions are also to be discussed. Summations and products of series are illustrated.

3.2.1 Taylor series expansion

Taylor series expansion of single-variable functions

The Taylor series expansion about the point $x = 0$ can be written as

$$f(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_k x^{k-1} + o(x^k) \tag{3.9}$$

where the coefficients a_i can be obtained from

$$a_i = \frac{1}{i!} \lim_{x\to 0} \frac{\mathrm{d}^{i-1}}{\mathrm{d}x^{i-1}} f(x), \quad i = 1, 2, 3, \cdots. \tag{3.10}$$

The expansion is also referred to as the *Maclaurin series*. If the Taylor series expansion is made about the $x = a$ point, the series can then be written as

$$f(x) = b_1 + b_2(x-a) + b_3(x-a)^2 + \cdots + b_k(x-a)^{k-1} + o[(x-a)^k] \tag{3.11}$$

where the b_i coefficients can be obtained from

$$b_i = \frac{1}{i!} \lim_{x\to a} \frac{\mathrm{d}^{i-1}}{\mathrm{d}x^{i-1}} f(x), \quad i = 1, 2, 3, \cdots. \tag{3.12}$$

Taylor series expansion can be obtained by the use of the `taylor()` function, provided in the Symbolic Math Toolbox. The syntaxes of the function are

```
taylor(fun,x,k)      % Taylor series about x = 0 point
taylor(fun,x,k,a)    % Taylor series expansion about the x = a point
```

where *fun* is a symbolic expression of the original function and x is the independent variable. If there is only one independent variable in *fun*, x can be omitted. The argument k is the number of terms required in the expansion, with a default number of terms of 6. If an extra argument a is given, the expansion is then made about the $x = a$ point. The Taylor series expansion solutions are demonstrated in the following examples.

Example 3.17 Consider again the function $f(x) = \sin x/(x^2 + 4x + 3)$ given in Example 3.4. Find the first 9 terms of Taylor series expansion about $x = 0$ point. Consider also the series expansions about points $x = 2$ and $x = a$.

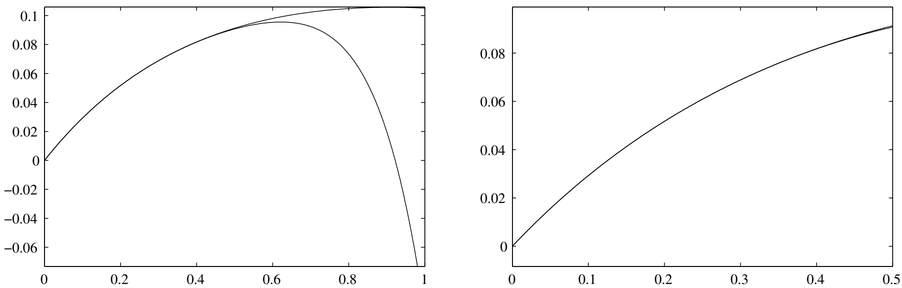
Solution The following statements can be used to specify the given function. The first 9 terms of Taylor series expansion can be obtained using the following statements

```
>> syms x; f=sin(x)/(x^2+4*x+3); y1=taylor(f,x,9)
```

and the result is

$$f(x) = \frac{1}{3}x - \frac{4}{9}x^2 + \frac{23}{54}x^3 - \frac{34}{81}x^4 + \frac{4087}{9720}x^5 - \frac{3067}{7290}x^6 + \frac{515273}{1224720}x^7 - \frac{386459}{918540}x^8 + \cdots.$$

In classical calculus courses, no analysis had been made upon the fitting quality of the finite number of terms approximation for a given function, since there were no ready tools available. With the use of MATLAB, the original function as well as the finite term Taylor series approximation can be comapred graphically as shown in Figure 3.4 (a).



(a) over interval $(0,1)$ (b) interval $(0,0.5)$

FIGURE 3.4: Finite term Taylor series approximation

```
>> ezplot(f,[0,5]), hold on; ezplot(y1,[0,5])
```

It can be seen that the fitting over the specified interval (0,1) is not satisfactory in the sense that when t is close to 1, the fitting is very poor. Thus 9 terms of Taylor series expansion for the original function are not enough. If the interval is changed to $(0,0.5)$, the fitting quality is shown in Figure 3.4 (b) which is good enough. Thus with the graphical facilities in MATLAB, the fitting qualities can be examined easily.

Now consider the Taylor series expansion about the point $x=2$. The series can be derived using the following statement:

```
>> taylor(f,x,9,2)
```

Since the expansion is lengthy, only first five terms are shown here

$$f(x) \approx \frac{\sin 2}{15}+\left(\frac{\cos 2}{15}-\frac{8\sin 2}{225}\right)(x-2)-\left(\frac{127\sin 2}{6750}+\frac{8\cos 2}{225}\right)(x-2)^2$$
$$+\left(\frac{23\cos 2}{6750}+\frac{628\sin 2}{50625}\right)(x-2)^3+\left(-\frac{15697}{6075000}\sin(2)+\frac{28}{50625}\cos(2)\right)(x-2)^4.$$

If one wants to find the series expansion about the $x=a$ point, Taylor series expansion can still be derived using similar statements

```
>> syms a; taylor(f,x,9,a)
```

Here only the first three terms are shown

$$\frac{\sin a}{a^2+3+4a}+\left[\frac{\cos a}{a^2+3+4a}-\frac{(4+2a)\sin a}{(a^2+3+4a)^2}\right](x-a)+\left[-\frac{\sin a}{(a^2+3+4a)^2}\right.$$
$$\left.-\frac{\sin a}{2(a^2+3+4a)}-\frac{\left(a^2\cos a+3\cos a+4a\cos a-4\sin a-2a\sin a\right)(4+2a)}{(a^2+3+4a)^3}\right](x-a)^2.$$

Example 3.18 Expand the sinusoidal function $y=\sin x$ into Taylor series, and compare the approximation quality for different terms.

Solution In order to find out the relationship between the fitting quality and the number of terms used, the loop structure should be used. The following statements can be issued to solve the problem, where the fitting curves shown in Figure 3.5 can be obtained.

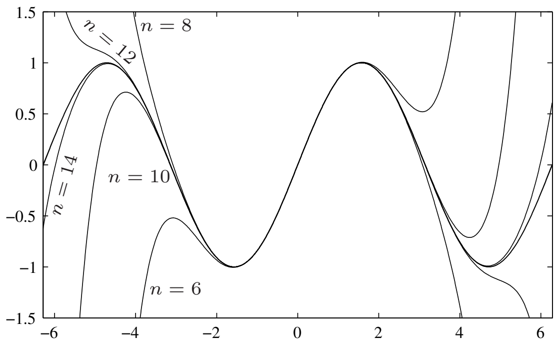


FIGURE 3.5: Taylor series approximation to given sinusoidal functions

```
>> x0=-2*pi:0.01:2*pi; y0=sin(x0); syms x; y=sin(x);
   plot(x0,y0), axis([-2*pi,2*pi,-1.5,1.5]);
   for n=[8:2:20]
      p=taylor(y,x,n), y1=subs(p,x,x0); line(x0,y1)
   end
```

For fewer terms, the satisfactory fitting interval is small. If the number of terms is increased, the satisfactory fitting interval will also increase. For instance, if one selects $n = 16$, the fitting is satisfactory over the interval $(-2\pi, 2\pi)$. The first 20 terms in the Taylor series expansion are obtained as

$$\sin x \approx x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9 - \frac{1}{39916800}x^{11} + \frac{1}{6227020800}x^{13} - \frac{1}{1307674368000}x^{15} + \frac{1}{355687428096000}x^{17} - \frac{1}{121645100408832000}x^{19}.$$

Taylor series expansion of multivariable functions

The Taylor series expansion of a multivariable function $f(x_1, x_2, \cdots, x_n)$ is

$$\begin{aligned} f(x_1, \cdots, x_n) = {} & f(a_1, \cdots, a_n) + \\ & \left[(x_1 - a_1)\frac{\partial}{\partial x_1} + \cdots + (x_n - a_n)\frac{\partial}{\partial x_n}\right] f(a_1, \cdots, a_n) + \\ & \frac{1}{2!}\left[(x_1 - a_1)\frac{\partial}{\partial x_1} + \cdots + (x_n - a_n)\frac{\partial}{\partial x_n}\right]^2 f(a_1, \cdots, a_n) + \cdots + \\ & \frac{1}{k!}\left[(x_1 - a_1)\frac{\partial}{\partial x_1} + \cdots + (x_n - a_n)\frac{\partial}{\partial x_n}\right]^k f(a_1, \cdots, a_n) + \cdots, \end{aligned} \tag{3.13}$$

where $(a_1, \cdots, a_n)$ is the center point of Taylor series expansion. In order to avoid misunderstanding, the terms can be regarded as the derivatives of function *fun*. Then the function evaluation can be made to the point $(a_1, a_2, \cdots, a_n)$. There is no existing function provided in the Symbolic Math Toolbox of MATLAB. However, the `mtaylor()` function in Maple can be used instead. The Taylor series expansion to multivariable functions can be obtained from

```
F=maple('mtaylor',fun,'[x1,··· ,xn]',k)              % about the origin
F=maple('mtaylor',fun,'[x1=a1,··· ,xn=an]',k)  % about (a1,··· ,an)
```

where $k-1$ is the highest degree in the expansion, and *fun* is the multivariable function. It should be noted that the quotation marks cannot be omitted since the information within the quotation marks will be passed to the Maple function directly.

Example 3.19 Consider again the function $z = f(x, y) = (x^2 - 2x)\mathrm{e}^{-x^2-y^2-xy}$ shown in Example 3.5. Find its Taylor series expansion.

Solution The following statements can be used to get the Taylor series expansion about the origin

```
>> syms x y; f=(x^2-2*x)*exp(-x^2-y^2-x*y);
   F=maple('mtaylor',f,'[x,y]',9); collect(F,x) % collect the polynomial
```

whose mathematical representation is

$$f(x,y) = -\frac{1}{6}x^8 + \left(-\frac{1}{2}y + \frac{1}{3}\right)x^7 + \left(-y^2 + y + \frac{1}{2}\right)x^6 + \left(-\frac{7}{6}y^3 + 2y^2 - 1 + y\right)x^5$$
$$+ \left(-y^4 - 1 - 2y + \frac{7}{3}y^3 + \frac{3}{2}y^2\right)x^4 + \left(2 + 2y^4 - y - \frac{1}{2}y^5 + y^3 - 3y^2\right)x^3$$
$$+ \left(2y + 1 + \frac{1}{2}y^4 - \frac{1}{6}y^6 - 2y^3 - y^2 + y^5\right)x^2 + \left(-2 - y^4 + 2y^2 + \frac{1}{3}y^6\right)x.$$

If one wants to expand the original function about $x = 1, y = a$ point, the following statements can be used

```
>> syms a; F=maple('mtaylor',f,'[x=1,y=a]',5);
```

and the expansion can be found as

$$f(x,y) = -\mathrm{e}^{-1-a-a^2} - \mathrm{e}^{-1-a-a^2}(-2-a)(x-1) - \mathrm{e}^{-1-a-a^2}(-2a-1)(y-a) +$$
$$\left[-\mathrm{e}^{-1-a-a^2}\left(1 + 2a + \frac{a^2}{2}\right) + \mathrm{e}^{-1-a-a^2}\right](x-1)^2 -$$
$$\mathrm{e}^{-1-a-a^2}\left(5a+1+2a^2\right)(y-a)(x-1) - \mathrm{e}^{-1-a-a^2}\left(-\frac{1}{2}+2a+2a^2\right)(y-a)^2 + \cdots.$$

In fact, the Maple function `mtaylor()` can also be used in evaluating the Taylor series expansion for single variable functions. The function call is almost as simple as `taylor()` function

```
>> F=maple('mtaylor',f,'[x=a]',5);
```

and the result obtained is

$$f(x,y) = (a^2-2a)\mathrm{e}^{-a^2-y^2-ay} + \left[(a^2-2a)(-2a-y) + (2a-2)\right]\mathrm{e}^{-a^2-y^2-ay}(x-a) +$$
$$\left[(a^2-2a)(-1+2a^2+2ay+y^2/2) + 1 + (2a-2)(-2a-y)\right]\mathrm{e}^{-a^2-y^2-ay}(x-a)^2 + \cdots.$$

3.2.2 Fourier series expansion

Consider a periodic function $f(x)$ defined over the interval $x \in [-L, L]$. The function is with a period of $T = 2L$. For the function defined on other intervals, it can be extended to periodic functions such that $f(x) = f(kT+x)$, where k is an arbitrary integer. A given function $f(x)$ can be expressed as an infinite series such that

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos \frac{n\pi}{L} x + b_n \sin \frac{n\pi}{L} x \right) \tag{3.14}$$

where

$$\begin{cases} a_n = \dfrac{1}{L} \displaystyle\int_{-L}^{L} f(x) \cos \frac{n\pi x}{L} \, \mathrm{d}x, & n = 0, 1, 2, \cdots \\ b_n = \dfrac{1}{L} \displaystyle\int_{-L}^{L} f(x) \sin \frac{n\pi x}{L} \, \mathrm{d}x, & n = 1, 2, 3, \cdots . \end{cases} \tag{3.15}$$

Such a series is referred to as the *Fourier series* and a_n, b_n are referred to as *Fourier coefficients*. If the function is defined over $x \in (a, b)$, it can be found that $L = (b-a)/2$. One may introduce a new variable $\hat{x}$ such that $x = \hat{x} + L + a$, the function $f(\hat{x})$ can be mapped into the symmetrical interval $(-L, L)$. Fourier series can be established for the new transformed function. Then the variable substitution $\hat{x} = x - L - a$ can be used to map the series back to the function of x.

There is no existing function for Fourier series expansion provided in MATLAB and Maple. Thus based on the above formula, the following MATLAB function can be prepared and placed in the `@sym` directory

```
function [A,B,F]=fseries(f,x,p,a,b)
if nargin==3, a=-pi; b=pi; end
L=(b-a)/2; if a+b, f=subs(f,x,x+L+a); end
A=int(f,x,-L,L)/L; B=[]; F=A/2;
for n=1:p
   an=int(f*cos(n*pi*x/L),x,-L,L)/L;
   bn=int(f*sin(n*pi*x/L),x,-L,L)/L; A=[A, an]; B=[B,bn];
   F=F+an*cos(n*pi*x/L)+bn*sin(n*pi*x/L);
end
if a+b, F=subs(F,x,x-L-a); end
```

The syntax of the function is `[`$\boldsymbol{A}$`,`$\boldsymbol{B}$`,`F`]=fseries(`f`,`x`,`p`,`a`,`b`)`, where f is the given function; x is the independent variable; p is number of the terms required in the expansion and (a, b) is the interval for x. If a, b arguments are omitted, the default interval $[-\pi, \pi]$ will be used. The returned arguments $\boldsymbol{A}$, $\boldsymbol{B}$ contain the Fourier coefficients, F is the Fourier series expansion obtained. Similar to the analytical function `fseries()`, the numerical version can also be written easily.

Example 3.20 Find the Fourier series expansion to the function $y = x(x-\pi)(x-2\pi)$, where $x \in (0, 2\pi)$.

Solution The Fourier series for the given function can easily be expressed

```
>> syms x; f=x*(x-pi)*(x-2*pi); [A,B,F]=fseries(f,x,12,0,2*pi)
```

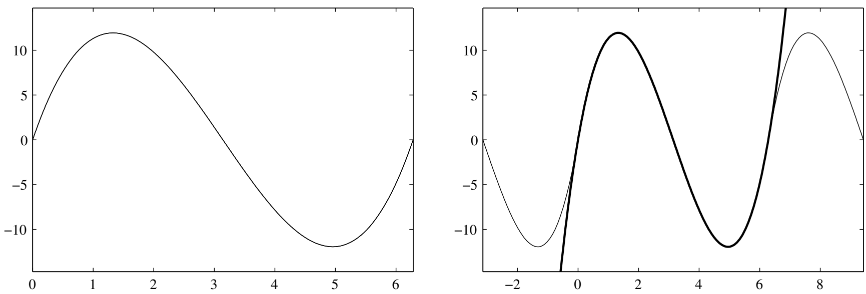
where the first 12 terms in the Fourier series are as follows:

$$f(x) = 12\sin x + \frac{3\sin 2x}{2} + \frac{4\sin 3x}{9} + \frac{3\sin 4x}{16} + \frac{12\sin 5x}{125} + \frac{\sin 6x}{18} + \frac{12\sin 7x}{343} + \frac{3\sin 8x}{128} + \frac{4\sin 9x}{243} + \frac{3\sin 10x}{250} + \frac{12\sin 11x}{1331} + \frac{\sin 12x}{144}.$$

From these results, the analytical form can be summarized as $f(x) = \sum_{n=1}^{\infty} \frac{12}{n^3}\sin nx$.

The first 12 terms in the Fourier series expansion and the original function can be graphically compared as shown in Figure 3.6 (a) with the following statements

```
>> ezplot(f,[0,2*pi]), hold on, ezplot(F,[0,2*pi])
```



(a) over interval $(0, 2\pi)$ (b) a larger interval $(-\pi, 3\pi)$

FIGURE 3.6: Accuracy of finite term Fourier series approximation

If one wants to further examine the approximation over a larger interval $x \in (-\pi, 3\pi)$, the following statements should be used

```
>> ezplot(f,[-pi,3*pi]), hold on, ezplot(F,[-pi,3*pi])
```

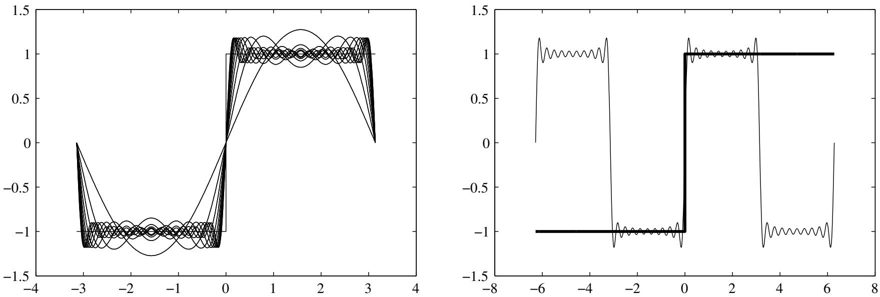
and the curves are shown in Figure 3.6 (b). It can be seen that over the $(0, 2\pi)$ interval the fitting is quite good. In other regions, since the Fourier series is made upon the assumption that it is periodically extended, thus it cannot approximate the original function in other intervals at all.

Example 3.21 Now consider a square wave defined over the interval $(-\pi, \pi)$, where $y = 1$ when $x \geqslant 0$, and $y = -1$ otherwise. Expand the function using Fourier series and observe how many terms in the function may give good approximation.

Solution Since in symbolic expressions inequality cannot be used, the square wave can be expressed as $f(x) = |x|/x$. In this way, the numerical and analytical expressions in Fourier series can be obtained for different terms in the expression. The curves can be obtained as shown in Figure 3.7 (a).

```
>> syms x; f=abs(x)/x;  % square wave definition
   xx=[-pi:pi/200:pi]; xx=xx(xx~=0); xx=sort([xx,-eps,eps]); % remove 0
   yy=subs(f,x,xx); plot(xx,yy), hold on  % draw the original function
```

```
for n=1:20
   [a,b,f1]=fseries(f,x,n); y1=subs(f1,x,xx); plot(xx,y1)
end
```



(a) over interval $(-\pi, \pi)$ (b) a larger interval $(-2\pi, 2\pi)$

FIGURE 3.7: Approximation of square wave by Fourier series

It can be seen that when 10 terms are used, the approximation is satisfactory. Even if the number of terms increases, the fitting accuracy may not be improved significantly. A finite Fourier series of the original function can be obtained by

```
>> [a,b,f1]=fseries(f,x,14); f1
```

and the expansion can be written as

$$f(x) \approx 4\frac{\sin x}{\pi} + \frac{4\sin 3x}{3\pi} + \frac{4\sin 5x}{5\pi} + \frac{4\sin 7x}{7\pi} + \frac{4\sin 9x}{9\pi} + \frac{4\sin 11x}{11\pi} + \frac{4\sin 13x}{13\pi}$$

which can further be summarized as $f(x) = \dfrac{4}{\pi}\displaystyle\sum_{k=1}^{\infty}\frac{\sin(2k-1)x}{2k-1}$.

Again the Fourier series expansion is established upon the assumption that it is periodically extended over the original function, thus the fitting in other intervals may be incorrect, as shown in Figure 3.7 (b).

```
>> xx=[-2*pi:pi/200:2*pi]; xx=xx(xx~=0); xx=sort([xx,-eps,eps]);
   yy=subs(f,x,xx); plot(xx,yy), y1=subs(f1,x,xx); line(xx,y1)
```

3.2.3 Series

The function `symsum()` provided in the Symbolic Math Toolbox can be used to evaluate the finite and infinite series with known general terms. The syntax of the function is S=`symsum(`f_k`,`k`,`k_0`,`k_n`)`, where f_k is the general term of the series, k is the independent term, and k_0 and k_n are the initial and final terms of the series, respectively. They can be set to `inf` for infinite series. The series can be written as

$$S = \sum_{k=k_0}^{k_n} f_k. \tag{3.16}$$

If there is only one independent variable defined in f_k, the variable k can be omitted in the function call.

Example 3.22 Compute the finite sum $S = 2^0 + 2^1 + 2^2 + \cdots + 2^{62} + 2^{63} = \sum_{i=0}^{63} 2^i$.

Solution Numerical solution to the problem can be found from

```
>> format long; s=sum(2.^[0:63])
```

with $s = 1.844674407370955 \times 10^{19}$. Since the data type of `double` is used, only 16 digits can be reserved. Thus the exact result cannot be obtained under double-precision scheme. The function `symsum()` can be used to solve the problem

```
>> syms k; symsum(2^k,0,63)
```

where $s_1 = 18446744073709551615$ can be obtained. The problem can even be solved with a simpler command `sum(sym(2).^[0:63])`, and the same result can be obtained. The method can be extended to calculate for more terms, for instance, it is possible to calculate the sum to 201 terms

```
>> s2=symsum(2^k,0,200)
```

and $s_2 = 3213876088517980551083924184682325205044405987565585670602751$. The exact solution cannot possibly be obtained using the double-precision data type.

Example 3.23 Compute the infinite series

$$S = \frac{1}{1 \times 4} + \frac{1}{4 \times 7} + \frac{1}{7 \times 10} + \cdots + \frac{1}{(3n-2)(3n+1)} + \cdots.$$

Solution With the use of the symbolic function

```
>> syms n; s=symsum(1/((3*n-2)*(3*n+1)),n,1,inf)
```

the sum result $s = 1/3$ can be obtained. The same problem can be tried using numerical method with `double` data type. For instance, if 10,000,000 terms are selected to be added up, the following statements can be used directly

```
>> m=1:10000000; s1=sum(1./((3*m-2).*(3*m+1))); format long; s1
```

and the sum is $s_1 = 0.33333332222165$. It can be seen that although a very large number of terms are selected with much long time consumed, there still exists unavoidable difference and the error reaches 10^{-6} level. It can be seen that when $m = 10^7$, the value of the general term is around 10^{-15}, thus it seems that the additional error in the summation may not be very large. In fact, since double-precision data type is used, some of the terms may not be added to the S variable. Thus even though more terms are used in the summation, the accuracy cannot be further increased.

Example 3.24 Evaluate the infinite series with an extra variable x.

$$J = 2 \sum_{n=0}^{\infty} \frac{1}{(2n+1)(2x+1)^{2n+1}}.$$

Solution In the examples studied earlier, numerical methods can be used to find the approximate solutions. If in the general term, extra independent variables are involved, numerical methods can no longer be used. Symbolic method has to be used to solve the problem. For instance, the sum can be evaluated with

```
>> syms n x; s1=symsum(2/((2*n+1)*(2*x+1)^(2*n+1)),n,0,inf);
   simple(s1)
```

and the infinite sum is $s_1 = \ln[(x+1)/x]$.

Example 3.25 Solve the limit problem with the series

$$\lim_{n\to\infty}\left[\left(1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}\right)-\ln n\right].$$

Solution So far, the series and limit problems have been discussed and illustrated separately. For this mixed problem, the following MATLAB statements can be used to solve it, where the finite sum should be made first using `symsum(1/m,m,1,n)`

```
>> syms m n; limit(symsum(1/m,m,1,n)-log(n),n,inf)
```

and `eulergamma` can be obtained, i.e., the Euler constant γ can be obtained whose value can be evaluated with `vpa(ans)` such that $\gamma = 0.5772156649015328606065120 9$.

It should be noted that in the computation, one should not evaluate the infinite sum before limit. Otherwise the original problem cannot be correctly solved.

3.2.4 Sequence product

The computation of sequence product $\prod_{n=a}^{b} f(n)$ is not directly supported in MATLAB. We can call Maple function `product()` from MATLAB to solve the product problem. The syntaxes of the function are

`maple('product(fun, n=a..b)')` or `maple('product',fun,'n=a..b')`

Example 3.26 Calculate the sequence product $\prod_{i=2}^{\infty}\left(1-\frac{2}{i(i+1)}\right)$.

Solution The sequence product is simply 1/3 .

```
>> syms i; maple('product',1-2/i/(i+1),'i=2..Inf')
```

3.3 Numerical Differentiation

If the original function is symbolically given, the analytical solutions to the differentiation problem can be obtained directly with the MATLAB built-in function `diff()`. The 100th order derivative can be obtained within seconds. However, in some applications where the original function is not known, only experimental data are given, the analytical or symbolic methods cannot be used. In this case, numerical methods must be used to get the derivatives from the experimental data. There is no dedicated function available in solving numerical differentiation problems in MATLAB. Thus, simple numerical algorithms are presented in this section with detailed implementation of the algorithms together with examples on how to solve the numerical differentiation problems in MATLAB.

3.3.1 Numerical differentiation algorithms

Assume that there is a set of measured data (t_i, y_i) with evenly distributed time instances $t_i = i\Delta t$, $i = 1, \cdots, N$, and the sampling period is Δt. The approximate derivative of the function can be defined as

$$y_i' \approx \frac{\Delta y_i}{\Delta t}; \quad y_i' = \frac{y_{i+1} - y_i}{\Delta t} + o(\Delta t). \tag{3.17}$$

This formula is also referred to as the *forward difference algorithm.*

Similarly, *backward difference formula* is defined as

$$y_i' \approx \frac{\Delta y_i}{\Delta t}; \quad y_i' = \frac{y_i - y_{i-1}}{\Delta t} + o(\Delta t). \tag{3.18}$$

From calculus, it is known that when $\Delta t \to 0$, the forward and backward formula can be used to solve analytically the differentiation problem. However, unfortunately, in practical applications, the condition $\Delta t \to 0$ cannot be satisfied. When the value of Δt is large, the accuracy of the differentiation cannot be guaranteed. So other improved numerical differentiation algorithms should be considered. For instance, the *central-point algorithm* can be used. The first-order derivative can also be defined as

$$y_i' \approx \frac{\Delta y_i}{\Delta t}; \quad y_i' = \frac{y_{i+1} - y_{i-1}}{2\Delta t}. \tag{3.19}$$

Denote $\widetilde{f}'(x) = \dfrac{f(x+\Delta t) - f(x-\Delta t)}{2\Delta t}$. From Taylor series expansion, the above method can further be written as

$$\begin{aligned}\widetilde{f}'(x) = {} & \frac{f(x) + \Delta t f'(x) + \Delta t^2 f''(x)/2! + \Delta t^3 f''(\xi)/3! + o(\Delta t^4)}{2\Delta t} - \\ & \frac{f(x) - \Delta t f'(x) + \Delta t^2 f''(x)/2! - \Delta t^3 f''(\xi)/3! + o(\Delta t^4)}{2\Delta t} \\ = {} & f'(x) + \frac{\Delta t^3}{3!} f''(\xi).\end{aligned} \tag{3.20}$$

It can be shown that the precision of the numerical approximation algorithm of first order differentiation is $o(\Delta t^2)$. High-order differentiation formulae can be similarly derived as follows:

$$\begin{aligned} y_i'' &\approx \frac{y_{i+1}-2y_i+y_{i-1}}{\Delta t^2} \\ y_i''' &\approx \frac{y_{i+2}-2y_{i+1}+2y_{i-1}-y_{i-2}}{2\Delta t^3} \\ y_i^{(4)} &\approx \frac{y_{i+2}-4y_{i+1}+6y_i-4y_{i-1}+y_{i-2}}{\Delta t^4}. \end{aligned} \tag{3.21}$$

There is yet another set of central-point difference algorithms with even higher accuracy of $o(\Delta t^4)$, defined as follows:

$$\begin{aligned} y_i' &\approx \frac{-y_{i+2}+8y_{i+1}-8y_{i-1}+y_{i-2}}{12\Delta t} \\ y_i'' &\approx \frac{-y_{i+2}+16y_{i+1}-30y_i+16y_{i-1}-y_{i-2}}{12\Delta t^2} \\ y_i''' &\approx \frac{-y_{i+3}+8y_{i+2}-13y_{i+1}+13y_{i-1}-8y_{i-2}+y_{i-3}}{8\Delta t^3} \\ y_i^{(4)} &\approx \frac{-y_{i+3}+12y_{i+2}-39y_{i+1}+56y_i-39y_{i-1}+12y_{i-2}-y_{i-3}}{6\Delta t^4}. \end{aligned} \tag{3.22}$$

3.3.2 Central-point difference algorithm with MATLAB implementation

The numerical differentiation algorithm given in (3.22) has the error level of $o(\Delta t^4)$ which can be used to solve numerical differentiation problems with higher numerical accuracy. Even when Δt is not too small, good approximation can still be expected due to its error level. Based on the algorithm, a MATLAB function is prepared as follows:

```
function [dy,dx]=diff_ctr(y,Dt,n)
yx1=[y 0 0 0 0 0]; yx2=[0 y 0 0 0 0]; yx3=[0 0 y 0 0 0];
yx4=[0 0 0 y 0 0]; yx5=[0 0 0 0 y 0]; yx6=[0 0 0 0 0 y];
switch n
case 1
   dy = (-diff(yx1)+7*diff(yx2)+7*diff(yx3)-diff(yx4))/(12*Dt); L0=3;
case 2
   dy=(-diff(yx1)+15*diff(yx2)-15*diff(yx3)+diff(yx4))/(12*Dt^2);L0=3;
case 3
   dy=(-diff(yx1)+7*diff(yx2)-6*diff(yx3)-6*diff(yx4)+...
         7*diff(yx5)-diff(yx6))/(8*Dt^3); L0=5;
case 4
   dy = (-diff(yx1)+11*diff(yx2)-28*diff(yx3)+28*diff(yx4)-...
         11*diff(yx5)+diff(yx6))/(6*Dt^4);L0=5;
end
dy=dy(L0+1:end-L0); dx=([1:length(dy)]+L0-2-(n>2))*Dt;
```

The syntax of the function is `[`$\boldsymbol{d}_y$`,`$\boldsymbol{d}_x$`]=diff_ctr(`$\boldsymbol{y}$`,`Δt`,`n`)`, where $\boldsymbol{y}$ is the vector containing measured data for evenly distributed points, and Δt is the sampling period. The argument n specifies the order of derivatives. The returned arguments $\boldsymbol{d}_y$ is the derivative vector computed, while the argument $\boldsymbol{d}_x$ is the corresponding vector of independent variables. It should be noted that the two vectors are a few points shorter than the original $\boldsymbol{y}$ vector.

Example 3.27 The function defined in Example 3.4 is still used in the demonstration of the algorithm. Since the original function is known, the analytical solution can be obtained for comparison. Sample data of the function can be generated from the function, and with the help of the data, the derivatives of the first- up to the fourth-order can be calculated and the results can be compared with the analytical solutions.

Solution An evenly spaced vector $\boldsymbol{x}$ is generated first. Since the original function is known, the analytical solutions to derivatives can be obtained. Then, if one substitutes the vector $\boldsymbol{x}$ into the obtained analytical functions, the theoretical derivative vectors can be obtained for comparison.

```
>> h=0.05; x=0:h:pi; syms x1; y=sin(x1)/(x1^2+4*x1+3);
   yy1=diff(y); f1=subs(yy1,x1,x); % get the contrast data analytically
   yy2=diff(yy1); f2=subs(yy2,x1,x); yy3=diff(yy2); f3=subs(yy3,x1,x);
   yy4=diff(yy3); f4=subs(yy4,x1,x);
```

From the data points y_i generated above, the first-order up to the fourth-order derivatives from the data can be calculated easily with the function `diff_ctr()` and the results are shown in Figure 3.8, together with the exact solutions. It can be seen that one may not observe the difference.

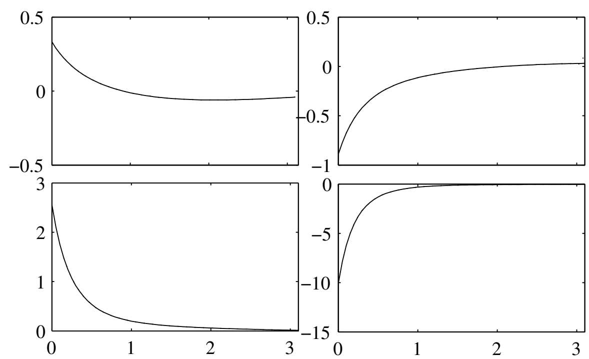


FIGURE 3.8: Comparisons of derivatives of different orders

```
>> y=sin(x)./(x.^2+4*x+3);    % generate the data to be used
   [y1,dx1]=diff_ctr(y,h,1); subplot(221), plot(x,f1,dx1,y1,':');
   [y2,dx2]=diff_ctr(y,h,2); subplot(222), plot(x,f2,dx2,y2,':')
   [y3,dx3]=diff_ctr(y,h,3); subplot(223), plot(x,f3,dx3,y3,':');
   [y4,dx4]=diff_ctr(y,h,4); subplot(224), plot(x,f4,dx4,y4,':')
```

Quantitative studies for the fourth-order derivative show that the maximum error between the exact results and the calculated results is as small as 3.5025×10^{-4}.

```
>> norm((y4-f4(4:60))./f4(4:60))
```

3.3.3 Gradient computations of functions with two variables

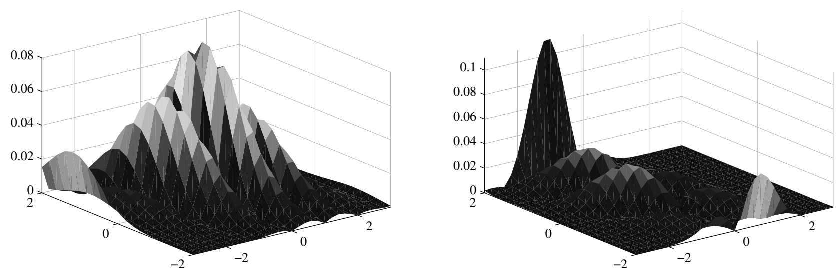
Consider the function $z(x, y)$ with two variables representing a 3D surface. The function `gradient()` can be used to calculate the gradients for the function. The syntax of the function is $[\boldsymbol{f}_x, \boldsymbol{f}_y]$=`gradient`$(\boldsymbol{z})$, where the "gradients" $\boldsymbol{f}_x$ and $\boldsymbol{f}_y$ thus calculated are not the actual gradients, since the coordinates x and y are not considered. If the matrix $\boldsymbol{z}$ is obtained, the gradients can be obtained using the following statements $\boldsymbol{f}_x=\boldsymbol{f}_x/\Delta x$, $\boldsymbol{f}_y=\boldsymbol{f}_y/\Delta y$, where Δx and Δy are respectively the step-sizes for x and y.

Example 3.28 Consider the function given in Example 3.5. Assume that the mesh grid data can be generated. Compute the gradients of the original function and analyze the error.

Solution The data can be generated using the following statements. The gradients here are obtained from the data rather than from the analytical function. The 3D attractive curves can also be drawn as shown in Figure 3.9 and it should be the same as the one in Figure 3.3 (b).

```
>> syms x y; z=(x^2-2*x)*exp(-x^2-y^2-x*y);
   [x0,y0]=meshgrid(-3:.2:3,-2:.2:2); z0=subs(z,{x,y},{x0,y0});
   [fx,fy]=gradient(z0); fx=fx/0.2; fy=fy/0.2;
   contour(x0,y0,z0,30); hold on; quiver(x0,y0,fx,fy)
```

The error surface is shown in Figure 3.9 where it can be seen that in most regions, the errors are relatively small. In other areas, the errors are large. This means that the spacing in the grid is too large to provide accurate gradient information. In order to reduce the error, the step-size should be reduced.

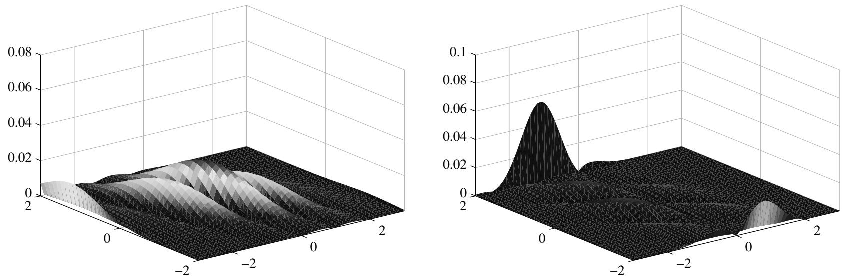


(a) $\partial z/\partial x$ (b) $\partial z/\partial y$

FIGURE 3.9: Error surface of the gradient of function with two variables

```
>> zx=diff(z,x); zx0=subs(zx,{x,y},{x0,y0});
   zy=diff(z,y); zy0=subs(zy,{x,y},{x0,y0});
   surf(x0,y0,abs(fx-zx0)); axis([-3 3 -2 2 0,0.08])
   figure; surf(x0,y0,abs(fy-zy0)); axis([-3 3 -2 2 0,0.11])
```

If the spacing in grids is reduced both by half, the following statements can be used and the new error surface can be calculated again as shown in Figure 3.10. It can be observed that the error is also reduced compared to Figure 3.9.



(a) $\partial z/\partial x$ (b) $\partial z/\partial y$

FIGURE 3.10: The error surface with reduced spacing in mesh grids

```
>> [x1,y1]=meshgrid(-3:.1:3,-2:.1:2); z1=subs(z,{x,y},{x1,y1});
   [fx,fy]=gradient(z1); fx=fx/0.1; fy=fy/0.1;
   zx1=subs(zx,{x,y},{x1,y1}); zy1=subs(zy,{x,y},{x1,y1});
   surf(x1,y1,abs(fx-zx1)); axis([-3 3 -2 2 0,0.08])
   figure; surf(x1,y1,abs(fy-zy1)); axis([-3 3 -2 2 0,0.1])
```

3.4 Numerical Integration Problems

3.4.1 Numerical integration from given data using trapezoidal method

Definite integral of the function with a single variable is defined as

$$I = \int_a^b f(x)\,\mathrm{d}x. \tag{3.23}$$

It is known that if the integrand $f(x)$ is theoretically not integrable, even with the powerful computer program, the analytical solutions to the problem cannot be obtained. Thus numerical solutions to the problems should be pursued instead. Numerical computation of an integral of single-variable function is also known as *quadrature*. There are various numerical quadrature algorithms to solve the integration problem. The widely used algorithms include the trapezoidal method, the Simpson's algorithm, the Romberg's algorithm, etc. The basic idea of the algorithms is to divide the whole interval $[a, b]$ into several sub-intervals $[x_i, x_{i+1}]$, $i = 1, 2, \cdots, N$, where $x_1 = a$ and $x_{N+1} = b$. Then the integration problem can be converted to the summation problem as follows:

$$\int_a^b f(x)\,\mathrm{d}x = \sum_{i=1}^{N} \int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x = \sum_{i=1}^{N} \Delta f_i. \tag{3.24}$$

The easiest method is to use trapezoidal approximation to each sub-interval. The numerical integration can be obtained by the use of `trapz()` function, whose syntax is `S=trapz(x,y)`, where $\boldsymbol{x}$ is a vector, and the number of rows of matrix $\boldsymbol{y}$ equals the number of the elements in vector $\boldsymbol{x}$. If the variable $\boldsymbol{y}$ is given as a multi-column matrix, the numerical integration to several functions can be evaluated simultaneously.

Example 3.29 Compute the definite integrals to the functions $\sin x, \cos x, \sin x/2$ within the interval $x \in (0, \pi)$ using the trapezoidal algorithm.

Solution The vector for horizontal axis is generated first and from it, the values of different functions can be evaluated such that the numerical integration can be obtained

```
>> x1=[0:pi/30:pi]'; y=[sin(x1) cos(x1) sin(x1/2)]; S=trapz(x1,y)
```

and the results are $S = [1.99817196134365, 0, 1.99954305299081]$.

Since the step-size is selected as $h = \pi/30 \approx 0.1$ which is considered as quite large, there exist errors in the results. In Section 8.1.2, the algorithm will be used with interpolation method to improve the quality of numerical integration results.

Example 3.30 Compute $\displaystyle\int_0^{3\pi/2} \cos 15x\,\mathrm{d}x$ with various step-sizes.

Solution Before solving the problem, the following statements can be used to draw the curves of the integrand as shown in Figure 3.11. It can be seen that there exists strong oscillation in the integrand.

```
>> x=[0:0.01:3*pi/2, 3*pi/2]; % the vector is assigned to ensure that
   y=cos(15*x); plot(x,y)      % the 3π/2 point is included
```

The theoretical solution to the problem is 1/15. For different step-sizes, $h =$ 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, the following statements can be used in solving approximately the integrals. The relevant results are given in Table 3.1.

```
>> syms x, A=int(cos(15*x),0,3*pi/2)
   h0=[0.1,0.01,0.001,0.0001,0.00001,0.000001]; v=[]; H=3*pi/2;
   for h=h0,
      x=[0:h:H,H]; y=cos(15*x); I=trapz(x,y); v=[v; h,I,1/15-I];
   end
```

It can be seen that when the step-size h reduces, so does the integral accuracy. For instance, if the step-size is selected as $h = 10^{-6}$, 11 digits can be preserved in

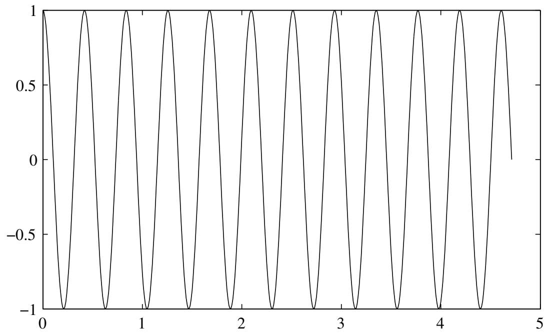


FIGURE 3.11: The plot of the integrand $f(x) = \cos 15x$

TABLE 3.1: Step-size selection and computation results

step	integral	error	time (s)	step	integral	error	time (s)
0.1	0.053891752	0.0127749	0.000	0.0001	0.066666654	1.25×10^{-8}	0.008
0.01	0.0665417	0.000125	0.005	10^{-5}	0.06666667	1.25×10^{-10}	0.033
0.001	0.066665417	1.25×10^{-6}	0.241	10^{-6}	0.066666667	1.25×10^{-12}	8.357

the result. Thus for this example, it takes as long as eight seconds for computation. If the step-size is further reduced, the computational effort demanded will be too high to be accepted.

3.4.2 Numerical integration of single variable functions

In traditional numerical analysis courses, several other numerical algorithms are usually explored for single variable functions. For instance, the approximate solutions Δf_i to the numerical integration problem can be solved with the Simpson's algorithm within the $[x_i, x_{i+1}]$ interval

$$\Delta f_i \approx \frac{h_i}{12}\left[f(x_i)+4f\left(x_i+\frac{h_i}{4}\right)+2f\left(x_i+\frac{h_i}{2}\right)+4f\left(x_i+\frac{3h_i}{4}\right)+f(x_i+h_i)\right] \tag{3.25}$$

where $h_i = x_{i+1} - x_i$. Based on the algorithm, a function `quad()` is provided in MATLAB to implement the variable-step-size Simpson's algorithm. The syntax of the function is

```
[y,k]=quad(fun,a,b)      % evaluate definite integral
[y,k]=quad(fun,a,b,ε)    % ibid with user-specified error
```

where *fun* can be used to specify the integrand. It can either be an M-file saved in `fun.m` file, or an anonymous function or an inline function. The syntax of such a function should be `y=fun(x)`. The arguments a and b are the lower- and upper-bounds in the definite integral, respectively. The argument ϵ is the user specified error tolerance, with a default value of 10^{-6}. The argument k returns the number of integrand function calls. With the information provided, the function `quad()` can be used directly to solve the numerical integration problem.

Example 3.31 For the integral $\operatorname{erf}(x) = \dfrac{2}{\sqrt{\pi}}\displaystyle\int_0^x \mathrm{e}^{-t^2}\mathrm{d}t$, which was shown not integrable, compute the integral using numerical methods.

Solution Before finding the numerical integration of a given function, the integrand should be specified first. There are three ways for specifying the integrand.

(i) **M-function** The first method is to express the integrand using a MATLAB function, where the input argument is the variable x. Since many x values need to be processed simultaneously, $\boldsymbol{x}$ vector can finally be used as the input argument, and the computation within the function should be expressed in dot operations. An example for expressing such a function is shown as

```
function y=c3ffun(x)
y=2/sqrt(pi)*exp(-x.^2);
```

The function can be saved as `c3ffun.m` file.

(ii) **Inline function** The integrand can also be described by the inline function, where the input argument $\boldsymbol{x}$ should be appended after the integrand expression. The integrand in this example can be written as

```
>> f=inline('2/sqrt(pi)*exp(-x.^2)','x');
```

(iii) **Anonymous function** Anonymous function expression is an effective way for describing the integrand. The format of the function is even more straightforward than the inline expression. The integrand can be expressed by the anonymous function as follows:

```
>> f=@(x)2/sqrt(pi)*exp(-x.^2);
```

It should be pointed out that the anonymous function expression is the fastest among the three. The drawbacks of the representation are that it can only return one argument, and function evaluations with intermediate computations are not allowed. Thus the anonymous function is used throughout the book whenever possible. If anonymous function cannot be used, the M-function description will be used.

When the integrand has been declared by any of the above three methods, the `quad()` function can be used to solve the definite integral problem

```
>> f=@(x)2/sqrt(pi)*exp(-x.^2); % anonymous function expression
   [I1,k1]=quad(f,0,1.5)         % evaluate the integration
   [I2,k2]=quad(@c3ffun,0,1.5)   % Alternatively M-function can be used
```

and $I_1 = I_2 = 0.96610518623173$, with 25 function calls. In fact, the high-precision solution to the same problem can be obtained with the use of Symbolic Math Toolbox

```
>> syms x, y0=vpa(int(2/sqrt(pi)*exp(-x^2),0,1.5),60)
```

where $y_0 = 0.966105146475310713936933729949905794996224943257461473285 75$.

Comparing the results obtained above, it can be found that the accuracy of the numerical method is not very high. This is due to the default setting of the error tolerance ϵ. One may reduce the value of ϵ to find solutions with higher accuracy. However, over-demanding in expected accuracy may lead to the failure of computation due to possible singularity problems

```
>> [y,k2]=quad(f,0,1.5,1e-20)     % high-precision is expected but failed
```

and a warning message is given, with an unreliable result $y = 0.96606$, $k_2 = 10009$.

```
Warning: Maximum function count exceeded; singularity likely.
  > In quad at 100
```

A new function `quadl()` is provided in MATLAB. The syntax of the function is exactly the same as the `quad()` function. The effective Lobatto algorithm is implemented in the function which is much more accurate than the `quad()` function.

Example 3.32 Consider the above example. Let us try to use the `quadl()` function to solve numerically the same problem and observe how the precision can be increased.

Solution Using `quadl()` function the following results can be obtained. Compared with the analytical solution, it can be found that the accuracy may reach 10^{-16} level. Although the pre-specified 10^{-20} error tolerance cannot be reached with double-precision computation, the solution is accurate enough for most applications

```
>> [y,k3]=quadl(f,0,1.5,1e-20), e=abs(y-y0)
```

and it can be found that $y = 0.96610514647531$, $e = 6.4\times 10^{-17}$, $k_3 = 1098$.

Example 3.33 Compute the integral of a piecewise function

$$I = \int_0^4 f(x)\,\mathrm{d}x, \text{ where } f(x) = \begin{cases} \mathrm{e}^{x^2}, & 0 \leqslant x \leqslant 2 \\ \dfrac{80}{4-\sin(16\pi x)}, & 2 < x \leqslant 4. \end{cases}$$

Solution The piecewise function is displayed in filled curve in Figure 3.12. It can be seen that the curve is not continuous at $x = 2$ point.

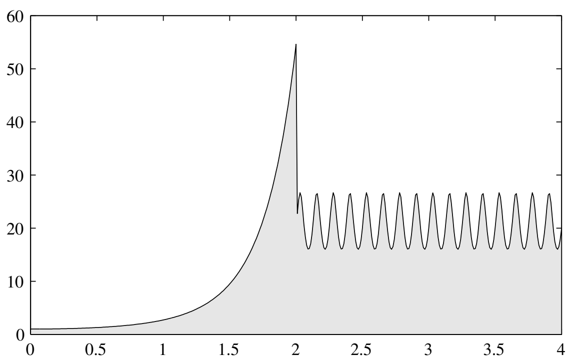


FIGURE 3.12: Filled plot of the integrand

```
>> x=[0:0.01:2, 2+eps:0.01:4,4];
   y=exp(x.^2).*(x<=2)+80./(4-sin(16*pi*x)).*(x>2);
   y(end)=0; x=[eps, x]; y=[0,y]; fill(x,y,'g')
```

With the use of relationship expressions, the integrand can be described and the functions quad() and quadl() can be used respectively to solve the original problem

```
>> f=@(x)exp(x.^2).*(x<=2)+80*(x>2)./(4-sin(16*pi*x));
   I1=quad(f,0,4), I2=quadl(f,0,4)
```

and it is found that $I_1 = 57.76435412500863$, $I_2 = 57.76445016946768$.

It can be seen from the obtained two results that there is a significant difference from the two methods. In fact, the original problem can also be divided into $\int_0^2 + \int_2^4$. The analytical solution function int() can then be used to find the analytical solutions to the original problem

```
>> syms x; I=vpa(int(exp(x^2),0,2)+int(80/(4-sin(16*pi*x)),2,4))
```

with $I = 57.764450125053010333315235385182$.

Compared with the analytical solutions, the results obtained by the quad() function may have large errors. If one divides the interval into two sub-intervals, there still exist errors. It can be concluded that quad() is not a good method to use. Let us try the quadl() function. Furthermore, the control options can be assigned such that even more accurate solutions can be obtained

```
>> f1=@(x)exp(x.^2); f2=@(x)80./(4-sin(16*pi*x));
   I1=quad(f1,0,2)+quad(f2,2,4), I2=quadl(f1,0,2)+quadl(f2,2,4)
   I3=quadl(f1,0,2,1e-11)+quadl(f2,2,4,1e-11)  % with error tolerance
```

the results are $I_1 = 57.764442889$, $I_2 = 57.76445012538$, and $I_3 = 57.764450125053$.

Example 3.34 Compute again the integral defined in Example 3.30.

Solution From the fixed-step algorithm demonstrated in Example 3.30, it can be found that only when the step-size is selected to be a very small value, the high accuracy can be achieved. However, with the help of variable-step algorithms, the original problem can be solved within a much shorter time and with much higher accuracy.

```
>> f=@(x)cos(15*x); tic, S=quadl(f,0,3*pi/2,1e-15), toc
```

It can be found that $S = 0.06666666666667$, and the elapsed time is 0.43 seconds, much faster than the fixed-step method. If quad() function is used, with error tolerance specified, the same warning and erroneous results are obtained.

Thus it can be concluded that the variable-step Lobatto algorithm of integrals has much more advantages over the fixed-step method taught in numerical analysis courses.

3.4.3 Numerical solutions to double integrals

Now consider the double integrals defined over a rectangular region

$$I = \int_{y_\mathrm{m}}^{y_\mathrm{M}} \int_{x_\mathrm{m}}^{x_\mathrm{M}} f(x,y)\,\mathrm{d}x\mathrm{d}y \tag{3.26}$$

and the function dblquad() can be used to solve this type of problem, with the syntaxes

y=`dblquad(`*fun*`,`x_m`,`x_M`,`y_m`,`y_M`)`	% double integral
y=`dblquad(`*fun*`,`x_m`,`x_M`,`y_m`,`y_M`,`ϵ`)`	% with given error tolerance

It should be noted that the number of calls to the integrand is not returned in this function. The users may set a global counter to check it when necessary.

Example 3.35 Compute the double definite integral

$$J = \int_{-1}^{1}\int_{-2}^{2} \mathrm{e}^{-x^2/2}\sin(x^2+y)\,\mathrm{d}x\mathrm{d}y.$$

Solution With the anonymous function to describe the integrand, the double integral can be evaluated numerically from the following statements

```
>> f=@(x,y)exp(-x.^2/2).*sin(x.^2+y);
   y=dblquad(f,-2,2,-1,1),
```

and the result is $y = 1.57456866245358$.

Unfortunately, the MATLAB function cannot be used in solving the double integral problem defined over a non-rectangular region as

$$I = \int_{x_\mathrm{m}}^{x_\mathrm{M}}\int_{y_\mathrm{m}(x)}^{y_\mathrm{M}(x)} f(x,y)\,\mathrm{d}y\mathrm{d}x. \tag{3.27}$$

A free toolbox, the Numerical Integration Toolbox (NIT) developed by Howard Wilson and Bryce Gardner can be downloaded from MathTools's website[1]. The function `gquad2dggen()` in the toolbox can be used to solve the numerical integration problem defined in (3.27). The syntaxes of the function are

J=`quad2dggen(`*fun*`,`F_lower`,`F_upper`,`x_m`,`x_M`)`	% double integral
J=`quad2dggen(`*fun*`,`F_lower`,`F_upper`,`x_m`,`x_M`,`ϵ`)`	% integral with error control

where ϵ is the error tolerance. This value can control the error in computation. However, if it is selected too small, significant amount of computational efforts will be required. In the function, three other MATLAB functions, i.e., the integrand and the inner upper- and lower-bound functions, are required to solve the problem.

It should be noted that the order of the integration is made with respect to x first then to y. If the user wants to integrate with respect to y first, the order x, y in the function should be changed first, before integration can be carried out. An illustrative example will be given to demonstrate this phenomenon.

Example 3.36 Compute the double definite integral

$$J = \int_{-1/2}^{1}\int_{-\sqrt{1-x^2/2}}^{\sqrt{1-x^2/2}} \mathrm{e}^{-x^2/2}\sin(x^2+y)\,\mathrm{d}y\mathrm{d}x.$$

[1]Download address: `http://www.mathtools.net/files/net/nittbx.zip`

Solution One can first integrate with respect to y, then to x, and then the inner bounds $y_{\rm M}(x)$ and $y_{\rm m}(x)$ can be defined. The following statements can then be used. Please note that the order of integration should be swaped first

```
>> fh=@(x)sqrt(1-x.^2/2);  fl=@(x)-sqrt(1-x.^2/2); % inner bounds
   f=@(y,x)exp(-x.^2/2).*sin(x.^2+y);              % order swaped
   y=quad2dggen(f,fl,fh,-1/2,1,eps),
```

and the value of integration can be found as $y = 0.41192954617630$. Now consider the analytical method

```
>> syms x y
   i1=int(exp(-x^2/2)*sin(x^2+y),y,-sqrt(1-x^2/2),sqrt(1-x^2/2));
   int(i1,x,-1/2,1)    % warning message given
```

and the following warning is displayed, after some time

```
Warning: Explicit integral could not be found.
> In sym.int at 58
ans =
int(2*exp(-1/2*x^2)*sin(x^2)*sin(1/2*(4-2*x^2)^(1/2)),x = -1/2..1)
```

and it can be seen that no explicit solution exists for this problem. One can get the high-precision numerical method with `vpa(ans,70)`, the result can be written as .4119295461762951196517599401760134672761827128252391627415959533602675.

It can be seen that the numerical solutions are very accurate.

If the original integral problem is changed to

$$J=\int_{-1}^{1}\int_{-\sqrt{1-y^2}}^{\sqrt{1-y^2}}{\rm e}^{-x^2/2}\sin(x^2+y)\,{\rm d}x{\rm d}y$$

the analytical problem cannot be used to find the results, even with the help of `vpa()` function. However, if the numerical method is used

```
>> fh=@(y)sqrt(1-y.^2); fl=@(y)-sqrt(1-y.^2); % inner bounds
   f=@(x,y)exp(-x.^2/2).*sin(x.^2+y);         % integrand
   I=quad2dggen(f,fl,fh,-1,1,eps),
```

which yields $I = 0.53686038269795$, which is different. For numerical methods, the numerical integration results will not be affected by whether the integrand is theoretically integrable or not.

3.4.4 Numerical solutions to triple integrals

The triple definite integral over the 3D rectangular region is given by

$$I=\int_{x_{\rm m}}^{x_{\rm M}}\int_{y_{\rm m}}^{y_{\rm M}}\int_{z_{\rm m}}^{z_{\rm M}}f(x,y,z)\,{\rm d}z{\rm d}y{\rm d}x; \tag{3.28}$$

the problem can be solved with the `triplequad()` function whose syntax is

I=`triplequad(`*fun*,$x_{\rm m}$,$x_{\rm M}$,$y_{\rm m}$,$y_{\rm M}$,$z_{\rm m}$,$z_{\rm M}$,ϵ,`@quadl)`

where *fun* describes the integrand. The argument ϵ can still be used in controlling the accuracy of the integration, with a default value of 10^{-6}. In order to increase the accuracy, smaller error tolerance can be assigned.

The extra function `@quadl` can be used to implement the integration for single variable functions. It can also be assigned to `@quad` or any other user functions.

Example 3.37 Compute the triple integral in Example 3.16

$$\int_0^2\int_0^\pi\int_0^\pi 4xz\mathrm{e}^{-x^2y-z^2}\mathrm{d}z\mathrm{d}y\mathrm{d}x.$$

Solution The anonymous function is used to specify the integrand. Thus the following statements can be used to compute the triple integral

```
>> f=@(x,y,z)4*x.*z.*exp(-x.*x.*y-z.*z);
   I=triplequad(f, 0,2, 0,pi, 0,pi, 1e-10,@quadl)
```

and $I = 3.108079402072966$.

NIT Toolbox can be used to solve multiple integral problems with other hyper-rectangular regions. For instance, the `quadndg()` function can be used for these problems. However, if the integration regions are not hyper rectangular regions, there are no existing implemented MATLAB functions available for numerical triple integrals.

3.5 Path Integrals and Line Integrals

Surprisingly, path integrals and line integrals cannot be solved by the existing MATLAB or Maple functions. In this section, the concepts and integration method for path and line integrals are summarized first and then solutions to these problems will be demonstrated through examples.

3.5.1 Path integrals

Path integrals are originated from the evaluation of the total mass of a spatial wire with unevenly distributed density. Assume that the density of a path l is $f(x, y, z)$. Then the total mass of the wire can be evaluated from the following equation

$$I_1 = \int_l f(x, y, z)\,\mathrm{d}s \tag{3.29}$$

where $\mathrm{d}s$ is the arc length at a certain point. Thus this kind of integral is also known as the *integral with respect to arc*. If $f(x, y, z) \equiv 1$, i.e., the density is evenly distributed and equals unity, the total length of the wire is calculated.

If the variables x, y and z are given respectively by parametric equations $x = x(t)$, $y = y(t)$, $z = z(t)$, they can be substituted into the $f(\cdot)$ function, and the differentiation of the arc can be written as

$$\mathrm{d}s = \sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}t}\right)^2 + \left(\frac{\mathrm{d}z}{\mathrm{d}t}\right)^2}\,\mathrm{d}t, \text{ or simply } \mathrm{d}s = \sqrt{x_t^2 + y_t^2 + z_t^2}\,\mathrm{d}t. \tag{3.30}$$

Then, the path integral can be converted into an ordinary integral with respect to t

$$I = \int_{t_\mathrm{m}}^{t_\mathrm{M}} f[x(t), y(t), z(t)]\sqrt{x_t^2 + y_t^2 + z_t^2}\,\mathrm{d}t. \tag{3.31}$$

For the integrand with two variables, $f(x, y)$, it can also be converted into ordinary integrals. Therefore, the solutions to the path integral problem can be solved with MATLAB using the previously described procedures.

Example 3.38 Compute $\displaystyle\int_l \frac{z^2}{x^2+y^2}\,\mathrm{d}s$, where the path l is defined as $x = a\cos t, y = a\sin t, z = at$, with $0 \leqslant t \leqslant 2\pi$ and $a > 0$.

Solution The following statements can be used for this path integral problem:

```
>> syms t; syms a positive; x=a*cos(t); y=a*sin(t); z=a*t;
   dx=diff(x,t); dy=diff(y,t); dz=diff(z,t);
   I=int(z^2/(x^2+y^2)*sqrt(dx^2+dy^2+dz^2),t,0,2*pi)
```

and the result is $I = \dfrac{8\sqrt{2}}{3}\pi^3 a$.

Example 3.39 Compute $\displaystyle\int_l (x^2 + y^2)\,\mathrm{d}s$ where path l is defined as the positive direction curve encircled by the paths $y = x$ and $y = x^2$.

Solution The following statements can be used to draw the two paths shown in Figure 3.13.

```
>> x=0:.001:1.2; y1=x; y2=x.^2; plot(x,y1,x,y2)
```

It can be seen that the original integration problem can be divided into two sub-integration problems. Thus the following statements can be used to add the two sub-integrations up to get the final solutions

```
>> syms x; y1=x; y2=x^2; I1=int((x^2+y2^2)*sqrt(1+diff(y2,x)^2),x,0,1);
   I2=int((x^2+y1^2)*sqrt(1+diff(y1,x)^2),x,1,0); I=I2+I1
```

and $I = -\dfrac{2}{3}\sqrt{2} + \dfrac{349}{768}\sqrt{5} + \dfrac{7}{512}\ln\left(-2+\sqrt{5}\right)$.

3.5.2 Line integrals

Line integral problems are originated from physics, where the total work is done by the force $\vec{f}(x, y, z)$ along a spatial curve l. This kind of integral problem can be expressed as

$$I_2 = \int_l \vec{\boldsymbol{f}}(x, y, z) \cdot \mathrm{d}\vec{\boldsymbol{s}} \tag{3.32}$$

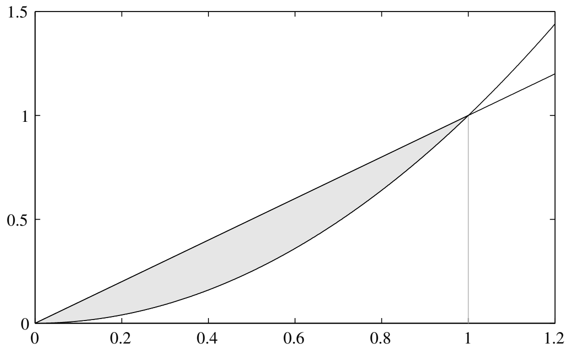


FIGURE 3.13: Illustration of the integration paths

where $\vec{\boldsymbol{f}}(x,y,z)=[P(x,y,z),Q(x,y,z),R(x,y,z)]$ is a row vector. The differentiation of the line $\mathrm{d}\vec{\boldsymbol{s}}$ is a column vector. If the line can be described by a parametric equation of t such as $x(t),y(t),z(t)$, with $t\in(a,b)$, the vector $\mathrm{d}\vec{\boldsymbol{s}}$ can then be written as

$$\mathrm{d}\vec{\boldsymbol{s}}=\left[\frac{\mathrm{d}x}{\mathrm{d}t},\frac{\mathrm{d}y}{\mathrm{d}t},\frac{\mathrm{d}z}{\mathrm{d}t}\right]^{\mathrm{T}}\mathrm{d}t. \tag{3.33}$$

The dot product of two vectors can be carried out directly and the line integrals can be re-defined as an ordinary integral as follows:

$$I_2=\int_a^b [P(x,y,z),Q(x,y,z),R(x,y,z)]\left[\frac{\mathrm{d}x}{\mathrm{d}t},\frac{\mathrm{d}y}{\mathrm{d}t},\frac{\mathrm{d}z}{\mathrm{d}t}\right]^{\mathrm{T}}\mathrm{d}t \tag{3.34}$$

which can be solved by using MATLAB.

Example 3.40 Compute the integral $\displaystyle\int_l \frac{x+y}{x^2+y^2}\,\mathrm{d}x-\frac{x-y}{x^2+y^2}\,\mathrm{d}y$, where the line l is defined as the positive circle given by $x^2+y^2=a^2$, $a>0$.

Solution If one wants to evaluate the line integral, the circle can be interpreted as the parametric equations $x=a\cos(t)$, $y=a\sin(t)$ for $0\leqslant t\leqslant 2\pi$. Thus the following statements can be used to calculate the line integral, with the result $I=2\pi$.

```
>> syms t; syms a positive; x=a*cos(t); y=a*sin(t);
   F=[(x+y)/(x^2+y^2),-(x-y)/(x^2+y^2)]; ds=[diff(x,t);diff(y,t)];
   I=int(F*ds,t,2*pi,0)  % positive circle, clockwise, t from 2π to 0
```

Example 3.41 Compute the line integral $\displaystyle\int_l (x^2-2xy)\,\mathrm{d}x+(y^2-2xy)\,\mathrm{d}y$, where the line l is defined as the parabolic curve $y=x^2$ $(-1\leqslant x\leqslant 1)$.

Solution In fact, the equations given are already the parametric equations of x. The derivative of x with respective to x is 1. The following statements can be used to solve the line integral problem, with the result $I=-14/15$.

```
>> syms x; y=x^2; F=[x^2-2*x*y,y^2-2*x*y]; ds=[1; diff(y,x)];
   I=int(F*ds,x,-1,1)
```

3.6 Surface Integrals

Two types of surface integrals are considered in this section, the scalar type and the vector type. The definitions and solutions to the problems will be summarized first followed by the detailed solution procedures with MATLAB script-based examples.

3.6.1 Scalar surface integrals

The scalar-type surface integrals are defined as

$$I = \iint_S \phi(x,y,z)\,\mathrm{d}S \tag{3.35}$$

where $\mathrm{d}S$ is the differentiated area. Thus this kind of integral is also referred to as the *surface integrals with respect to area*. If $\phi(x,y,z) \equiv 1$, the area of the surface can be computed.

Let the surface S be defined by $z = f(x,y)$. The original surface integral can be converted into a double integral over the x-y plane, such that

$$I = \iint_{\sigma_{xy}} \phi[x,y,f(x,y)]\sqrt{1+f_x^2+f_y^2}\,\mathrm{d}x\mathrm{d}y \tag{3.36}$$

where σ_{xy} is the integration region, which is an ordinary double integral problem.

Example 3.42 Compute $\displaystyle\iint_S xyz\mathrm{d}S$, where the integration surface S is defined as the region enclosed by the four planes $x=0$, $y=0$, $z=0$, and $x+y+z=a$, where $a>0$.

Solution Denote the four planes by S_1, S_2, S_3, and S_4. The original surface integral can be calculated using $\displaystyle\iint_S = \iint_{S_1} + \iint_{S_2} + \iint_{S_3} + \iint_{S_4}$. Considering the planes S_1, S_2, S_3, since the integrands are all 0, only the integral on the S_4 should be considered. The plane S_4 can mathematically be described as $z = a-x-y$. Then, the following statements can be used to evaluate the surface integral

```
>> syms x y; syms a positive; z=a-x-y;
   I=int(int(x*y*z*sqrt(1+diff(z,x)^2+diff(z,y)^2),y,0,a-x),x,0,a)
```

which gives $I = \sqrt{3}a^5/120$.

If the parametric equations for the surface are given by

$$x = x(u, v), \ y = y(u, v), \ z = z(u, v), \tag{3.37}$$

the surface integral can then be obtained using the following formula

$$I = \iint_{\Sigma} \phi[x(u, v), y(u, v), z(u, v)] \sqrt{EG - F^2} \, \mathrm{d}u\mathrm{d}v \tag{3.38}$$

where

$$E = x_u^2 + y_u^2 + z_u^2, \quad F = x_u x_v + y_u y_v + z_u z_v, \quad G = x_v^2 + y_v^2 + z_v^2. \tag{3.39}$$

Example 3.43 Compute the surface integral $\iint (x^2 y + z y^2) \, \mathrm{d}S$, where the surface S is defined as the surfaces composed of $x = u\cos v, y = u\sin v, z = v$, $0 \leqslant u \leqslant a, 0 \leqslant v \leqslant 2\pi$.

Solution The following statements can be used to calculate the integrals

```
>> syms u v; syms a positive;
   x=u*cos(v); y=u*sin(v); z=v; f=x^2*y+z*y^2;
   E=simple(diff(x,u)^2+diff(y,u)^2+diff(z,u)^2);
   F=diff(x,u)*diff(x,v)+diff(y,u)*diff(y,v)+diff(z,u)*diff(z,v);
   G=simple(diff(x,v)^2+diff(y,v)^2+diff(z,v)^2);
   I=int(int(f*sqrt(E*G-F^2),u,0,a),v,0,2*pi)
```

and the result is $I = \dfrac{1}{8}\pi^2 \left[2a\left(a^2+1\right)^{3/2} - a\sqrt{a^2+1} - \operatorname{arcsinh} a\right]$.

3.6.2 Vector surface integrals

The second category of surface integral is also referred to as the *surface integrals in vector fields*. Suppose the integrand is given by a row vector $\vec{\boldsymbol{\Gamma}} = [P, Q, R]$, while $\mathrm{d}\vec{\boldsymbol{V}}$ is given by a column vector $\mathrm{d}\vec{\boldsymbol{V}} = [\mathrm{d}y\mathrm{d}z, \mathrm{d}x\mathrm{d}z, \mathrm{d}x\mathrm{d}y]^{\mathrm{T}}$, the mathematical description to the problem is

$$I = \iint_{S^+} \vec{\boldsymbol{\Gamma}} \cdot \mathrm{d}\vec{\boldsymbol{V}} = \iint_{S^+} P(x, y, z) \, \mathrm{d}y\mathrm{d}z + Q(x, y, z) \, \mathrm{d}x\mathrm{d}z + R(x, y, z) \, \mathrm{d}x\mathrm{d}y, \tag{3.40}$$

where the positive surface S^+ is defined with $z = f(x, y)$. The surface integral problem can then be converted into the scalar surface integral problem

$$I = \iint_{S^+} \left[P(x, y, z) \cos\alpha + Q(x, y, z) \cos\beta + R(x, y, z) \cos\gamma\right] \, \mathrm{d}\boldsymbol{S} \tag{3.41}$$

where z is replaced by $f(x, y)$, and

$$\cos\alpha = \frac{-f_x}{\sqrt{1 + f_x^2 + f_y^2}}, \qquad \cos\beta = \frac{-f_y}{\sqrt{1 + f_x^2 + f_y^2}}, \qquad \cos\gamma = \frac{1}{\sqrt{1 + f_x^2 + f_y^2}}. \tag{3.42}$$

Thus, the $\sqrt{1+f_x^2+f_y^2}$ term may cancel the relevant term in (3.36), and the surface integral can be written as

$$I=\iint_{\sigma_{xy}} -Pf_x\,\mathrm{d}x\mathrm{d}y - Qf_y\,\mathrm{d}x\mathrm{d}z + R\,\mathrm{d}y\mathrm{d}z. \tag{3.43}$$

If the surface is described by the parametric equations in (3.37), the following equations can be obtained

$$\cos\alpha=\frac{A}{\sqrt{A^2+B^2+C^2}},\quad \cos\beta=\frac{B}{\sqrt{A^2+B^2+C^2}},\quad \cos\gamma=\frac{C}{\sqrt{A^2+B^2+C^2}} \tag{3.44}$$

where $A=y_uz_v-z_uy_v$, $B=z_ux_v-x_uz_v$, $C=x_uy_v-y_ux_v$. Then from the converted scalar surface integral (3.41), it can be found that the denominator in (3.44) cancels the $\sqrt{EG-F^2}$ term. Thus the vector surface integral can be simplified as the following standard double integral

$$I=\int_{v_\mathrm{m}}^{v_\mathrm{M}}\int_{u_\mathrm{m}(v)}^{u_\mathrm{M}(v)}[AP(u,v)+BQ(u,v)+CR(u,v)]\,\mathrm{d}u\mathrm{d}v. \tag{3.45}$$

Example 3.44 Compute the surface integral $\iint x^3\mathrm{d}y\mathrm{d}z$, where the surface S is defined as the positive side of the ellipsoid surface $\dfrac{x^2}{a^2}+\dfrac{y^2}{b^2}+\dfrac{z^2}{c^2}=1$.

Solution The parametric equations can be introduced such that $x=a\sin u\cos v$, $y=b\sin u\sin v$, $z=c\cos u$, and $\left(0\leqslant u\leqslant\dfrac{\pi}{2}\right)$, $(0\leqslant v\leqslant 2\pi)$. The following statements can be used to compute the surface integral, with the result $I=2\pi a^3cb/5$.

```
>> syms u v; syms a b c positive;
   x=a*sin(u)*cos(v); y=b*sin(u)*sin(v); z=c*cos(u);
   A=diff(y,u)*diff(z,v)-diff(z,u)*diff(y,v);
   I=int(int(x^3*A,u,0,pi/2),v,0,2*pi)
```

Exercises

1. Compute the following limit problems:
   (i) $\lim\limits_{x\to\infty}\left(3^x+9^x\right)^{\frac{1}{x}}$, (ii) $\lim\limits_{x\to\infty}\dfrac{(x+2)^{x+2}(x+3)^{x+3}}{(x+5)^{2x+5}}$
2. Compute the following double limit problems:
   (i) $\lim\limits_{\substack{x\to-1\\y\to2}}\dfrac{x^2y+xy^3}{(x+y)^3}$, (ii) $\lim\limits_{\substack{x\to0\\y\to0}}\dfrac{xy}{\sqrt{xy+1}-1}$, (iii) $\lim\limits_{\substack{x\to0\\y\to0}}\dfrac{1-\cos\left(x^2+y^2\right)}{\left(x^2+y^2\right)\mathrm{e}^{x^2+y^2}}$
3. Compute the derivatives of the following functions:

(i) $y(x) = \sqrt{x \sin x \sqrt{1 - \mathrm{e}^x}}$, (ii) $y(t) = \sqrt{\dfrac{(x-1)(x-2)}{(x-3)(x-4)}}$

(iii) $\operatorname{atan}\dfrac{y}{x} = \ln(x^2 + y^2)$, (iv) $y(x) = -\dfrac{1}{na}\ln\dfrac{x^n + a}{x^n}$, $n > 0$

4. Compute the 10th order derivative of the function $y = \dfrac{1 - \sqrt{\cos ax}}{x\left(1 - \cos\sqrt{ax}\right)}$.

5. In calculus courses, when the limit of a ratio is required, where both the numerator and the denominator tend to 0 or ∞, simultaneously, L'Hôpital's law can be used, i.e., to evaluate the limits of derivatives of numerator and denominator. Verify the $\lim\limits_{x\to 0}\dfrac{\ln(1+x)\ln(1-x) - \ln(1-x^2)}{x^4}$ by the consecutive use of L'Hôpital's law, and compare with the results directly obtained.

6. For parametric equation $\begin{cases} x = \ln\cos t \\ y = \cos t - t\sin t \end{cases}$, compute $\dfrac{\mathrm{d}y}{\mathrm{d}x}$ and $\left.\dfrac{\mathrm{d}^2y}{\mathrm{d}x^2}\right|_{t=\pi/3}$.

7. Assume that $u = \cos^{-1}\sqrt{\dfrac{x}{y}}$. Verify that $\dfrac{\partial^2 u}{\partial x \partial y} = \dfrac{\partial^2 u}{\partial y \partial x}$.

8. For a given function $\begin{cases} xu + yv = 0 \\ yu + xv = 1 \end{cases}$, compute $\dfrac{\partial^2 u}{\partial x \partial y}$.

9. Assume that $f(x,y) = \displaystyle\int_0^{xy} \mathrm{e}^{-t^2}\mathrm{d}t$. Compute $\dfrac{x}{y}\dfrac{\partial^2 f}{\partial x^2} - 2\dfrac{\partial^2 f}{\partial x \partial y} + \dfrac{\partial^2 f}{\partial y^2}$.

10. Given a matrix $\boldsymbol{f}(x,y,z) = \begin{bmatrix} 3x + \mathrm{e}^y z \\ x^3 + y^2 \sin z \end{bmatrix}$, compute its Jacobian matrix.

11. Compute the following infinite integrals:

(i) $I(x) = -\displaystyle\int \frac{3x^2 + a}{x^2\left(x^2+a\right)^2}\,\mathrm{d}x$, (ii) $I(x) = \displaystyle\int \frac{\sqrt{x(x+1)}}{\sqrt{x} + \sqrt{1+x}}\,\mathrm{d}x$

(iii) $I(x) = \displaystyle\int x\mathrm{e}^{ax}\cos bx\,\mathrm{d}x$, (iv) $I(t) = \displaystyle\int \mathrm{e}^{ax}\sin bx \sin cx\,\mathrm{d}x$

12. Compute the definite integrals and infinite integrals

(i) $I = \displaystyle\int_0^\infty \frac{\cos x}{\sqrt{x}}\,\mathrm{d}x$, (ii) $I = \displaystyle\int_0^1 \frac{1+x^2}{1+x^4}\,\mathrm{d}x$

13. For the function $f(x) = \mathrm{e}^{-5x}\sin(3x + \pi/3)$, compute $\displaystyle\int_0^t f(x)f(t+x)\,\mathrm{d}x$.

14. For different values of a, compute the integral $I = \displaystyle\int_0^\infty \frac{\cos ax}{1+x^2}\,\mathrm{d}x$.

15. Show that for any function $f(t)$, $\displaystyle\int_a^b f(t)\,\mathrm{d}t = -\int_b^a f(t)\,\mathrm{d}t$.

16. Solve the following multiple integral problems:

(i) $\displaystyle\int_0^2\int_0^1 \sqrt{4 - x^2 - y^2}\,\mathrm{d}y\mathrm{d}x$, (ii) $\displaystyle\int_0^3\int_0^{3-x}\int_0^{3-x-y} xyz\,\mathrm{d}z\mathrm{d}y\mathrm{d}x$

(iii) $\displaystyle\int_0^2\int_0^{\sqrt{4-x^2}}\int_0^{\sqrt{4-x^2-y^2}} z(x^2+y^2)\,\mathrm{d}z\mathrm{d}y\mathrm{d}x$

(iv) $\displaystyle\int_0^{7/10}\int_0^{4/5}\int_0^{9/10}\int_0^1\int_0^{11/10}\sqrt{6 - x^2 - y^2 - z^2 - w^2 - u^2}\,\mathrm{d}w\mathrm{d}u\mathrm{d}z\mathrm{d}y\mathrm{d}x$

17. Compute the Fourier series expansions for the following functions, and compare

graphically the approximation and exact results, using finite numbers of terms:

(i) $f(x)=(\pi-|x|)\sin x,\quad -\pi\leqslant x<\pi,$ (ii) $f(x)=\mathrm{e}^{|x|},\quad -\pi\leqslant x<\pi,$

(iii) $f(x)=\begin{cases}2x/l, & 0<x<l/2\\ 2(l-x)/l, & l/2<x<l\end{cases}$, where $l=\pi$.

18. Obtain the Taylor series expansions for the following functions, and compare graphically the approximation and exact results with finite numbers of terms:

(i) $\displaystyle\int_0^x \frac{\sin t}{t}\,\mathrm{d}t$, (ii) $\ln\left(\dfrac{1+x}{1-x}\right)$, (iii) $\ln\left(x+\sqrt{1+x^2}\right)$, (iv) $(1+4.2x^2)^{0.2}$,

(v) $\mathrm{e}^{-5x}\sin(3x+\pi/3)$ expansions about $x=0$ and $x=a$ points respectively.

(vi) $f(x,y)=\dfrac{1-\cos\left(x^2+y^2\right)}{\left(x^2+y^2\right)\mathrm{e}^{x^2+y^2}}$ expansion about $x=1, y=0$ point.

19. Compute the first n term finite sums and infinite sums.

(i) $\dfrac{1}{1\times 6}+\dfrac{1}{6\times 11}+\cdots+\dfrac{1}{(5n-4)(5n+1)}+\cdots$

(ii) $\left(\dfrac{1}{2}+\dfrac{1}{3}\right)+\left(\dfrac{1}{2^2}+\dfrac{1}{3^2}\right)+\cdots+\left(\dfrac{1}{2^n}+\dfrac{1}{3^n}\right)+\cdots$

20. Compute the following limits:

(i) $\displaystyle\lim_{n\to\infty}\left[\frac{1}{2^2-1}+\frac{1}{4^2-1}+\frac{1}{6^2-1}+\cdots+\frac{1}{(2n)^2-1}\right]$,

(ii) $\displaystyle\lim_{n\to\infty} n\left(\frac{1}{n^2+\pi}+\frac{1}{n^2+2\pi}+\frac{1}{n^2+3\pi}+\cdots+\frac{1}{n^2+n\pi}\right)$

21. Show that $\cos\theta+\cos 2\theta+\cdots+\cos n\theta=\dfrac{\sin(n\theta/2)\cos[(n+1)\theta/2]}{\sin\theta/2}$.

22. For the following tabulated measured data, evaluate numerically its derivatives and definite integral.

x_i	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2
y_i	0	2.2077	3.2058	3.4435	3.241	2.8164	2.311	1.8101	1.3602	0.9817	0.6791	0.4473	0.2768

23. Evaluate the definite integral $\displaystyle\int_0^{\pi}(\pi-t)^{\frac{1}{4}}f(t)\mathrm{d}t$, $f(t)=\mathrm{e}^{-t}\sin(3t+1)$ numerically.

Also evaluate the integration function $F(t)=\displaystyle\int_0^t (t-\tau)^{\frac{1}{4}}f(\tau)\,\mathrm{d}\tau$ numerically for different sample points of t, such that $t=0.1,0.2,\cdots,\pi$, and draw the $F(t)$ plot.

24. Evaluate numerically the following multiple integral problems. It should be noted that there are no analytical solutions to these problems. Therefore, the obtained numerical results should be double-checked by varying step sizes or default accuracies.

(i) $\displaystyle\int_0^2\int_0^{\mathrm{e}^{-x^2/2}}\sqrt{4-x^2-y^2}\ \mathrm{e}^{-x^2-y^2}\,\mathrm{d}y\mathrm{d}x$

(ii) $\displaystyle\int_0^2\int_0^{\sqrt{4-x^2}}\int_0^{\sqrt{4-x^2-y^2}} z(x^2+y^2)\,\mathrm{e}^{-x^2-y^2-z^2-xz}\,\mathrm{d}z\mathrm{d}y\mathrm{d}x$

(iii) $\displaystyle\int_0^1\int_0^2\int_0^{xy}\int_0^u \mathrm{e}^{6-x^2-y^2-z^2-u^2}\,\mathrm{d}u\mathrm{d}z\mathrm{d}y\mathrm{d}x$

25. Compute the gradient of the measured data for a function of two variables. Assume that the data were generated by the function $f(x, y) = 4 - x^2 - y^2$. Generate the data and verify the results of gradient with theoretical results.

0	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8	2
0	4	3.96	3.84	3.64	3.36	3	2.56	2.04	1.44	0.76	0
0.2	3.96	3.92	3.8	3.6	3.32	2.96	2.52	2	1.4	0.72	−0.04
0.4	3.84	3.8	3.68	3.48	3.2	2.84	2.4	1.88	1.28	0.6	−0.16
0.6	3.64	3.6	3.48	3.28	3	2.64	2.2	1.68	1.08	0.4	−0.36
0.8	3.36	3.32	3.2	3	2.72	2.36	1.92	1.4	0.8	0.12	−0.64
1	3	2.96	2.84	2.64	2.36	2	1.56	1.04	0.44	−0.24	−1
1.2	2.56	2.52	2.4	2.2	1.92	1.56	1.12	0.6	0	−0.68	−1.44
1.4	2.04	2	1.88	1.68	1.4	1.04	0.6	0.08	−0.52	−1.2	−1.96
1.6	1.44	1.4	1.28	1.08	0.8	0.44	0	−0.52	−1.12	−1.8	−2.56
1.8	0.76	0.72	0.6	0.4	0.12	−0.24	−0.68	−1.2	−1.8	−2.48	−3.24
2	0	−0.04	−0.16	−0.36	−0.64	−1	−1.44	−1.96	−2.56	−3.24	−4

26. Compute the following path and line integrals:

(i) $\displaystyle\int_l (x^2+y^2)\,\mathrm{d}s$, l: $x = a(\cos t + t\sin t), y = a(\sin t - t\cos t)$, for $0 \leqslant t \leqslant 2\pi$

(ii) $\displaystyle\int_l (yx^3+\mathrm{e}^y)\,\mathrm{d}x + (xy^3 + x\mathrm{e}^y - 2y)\,\mathrm{d}y$, where l is given by the upper-semi-ellipsis of $a^2x^2 + b^2y^2 = c^2$.

(iii) $\displaystyle\int_l y\,\mathrm{d}x - x\,\mathrm{d}y + (x^2+y^2)\,\mathrm{d}z$, l: $x = \mathrm{e}^t, y = \mathrm{e}^{-t}, z = at$, $0 \leqslant t \leqslant 1$, for $a > 0$.

(iv) $\displaystyle\int_l (\mathrm{e}^x\sin y - my)\mathrm{d}x + (\mathrm{e}^x\cos y - m)\,\mathrm{d}y$, where l is defined as the closed path from $(a, 0)$ to $(0, 0)$, then with the upper-semi-circle $x^2 + y^2 = ax$.

27. Compute the surface integrals, where S is the bottom side of the semi-sphere $z = \sqrt{R^2 - x^2 - y^2}$.

(i) $\displaystyle\int_S xyz^3\,\mathrm{d}s$, (ii) $\displaystyle\int_S (x + yz^3)\,\mathrm{d}x\mathrm{d}y$.

References and Bibliography

[1] Garbow B S, Boyle J M, Dongarra J J, et al. Matrix eigensystem routines — EISPACK guide extension, Lecture notes in computer sciences, volume 51. New York: Springer-Verlag, 1977

[2] Smith B T, Boyle J M, Dongarra J J, et al. Matrix eigensystem routines – EISPACK guide, Lecture notes in computer sciences, volume 6. New York: Springer-Verlag, second edition, 1976

[3] Dongarra J J, Bunsh J R, Moler C B. LINPACK user's guide. Philadelphia: Society of Industrial and Applied Mathematics, 1979

[4] Press W H, Flannery B P, Teukolsky S A, et al. Numerical recipes, the art of scientific computing. Cambridge: Cambridge University Press, 1986. Free textbook at http://www.nrbook.com/a/bookcpdf.php

[5] Lamport L. LaTeX: a document preparation system — user's guide and reference manual. Reading MA: Addision-Wesley Publishing Company, second edition, 1994

[6] Xue D. Analysis and computer aided design of nonlinear systems with Gaussian inputs. D.Phil. thesis, Sussex University, U.K., 1992

[7] Strang G. Calculus. Free textbook at http://ocw.mit.edu/ans7870/resources/Strang/strangtext.htm: Wellesley-Cambridge Press, 1991

[8] Dawkins P. Calculus I, II, & III. http://tutorial.math.lamar.edu/pdf/CalcII/CalcI_Complete.pdf; http://tutorial.math.lamar.edu/pdf/CalcII/CalcII_Complete.pdf; http://tutorial.math.lamar.edu/pdf/CalcIII/CalcIII_Complete.pdf, 2007

[9] Hefferon J. Linear algebra. Saint Michael's College, USA: Open source textbook at http://joshua.smcvt.edu/linearalgebra/, 2006

[10] Meyer C D. Matrix analysis and applied linear algebra. Philadelphia: Society for Industrial and Applied Mathematics. Free at http://www.matrixanalysis.com/DownloadChapters.html, 2001

[11] Dawkins P. Linear algebra. http://tutorial.math.lamar.edu/pdf/LinAlg/LinAlg_Complete.pdf, 2007

[12] Moler C B, Van Loan C F. Nineteen dubious ways to compute the exponential of a matrix. SIAM Review, 1979, 20:801–836

[13] Huang L. Linear algebra in systems and control theory. Beijing: Science Press, 1984 (in Chinese)

[14] Mauch S. Advanced mathematical methods for scientists and engineers. Open source textbook at http://www.its.caltech.edu/~sean/ applied_math.pdf, 2004

[15] Boyd S, Vandenberghe L. Convex optimization. Cambridge University Press, 2004. Free textbook at `http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`

[16] Boyd S, Ghaoui L El, Feron E, et al. Linear matrix inequalities in systems and control theory. Philadelphia: SIAM books, Volume 15 of Studies in Applied Mathematics. Free textbook at `http://www.stanford.edu/~boyd/lmibook/lmibook.pdf`, 1994

[17] Mittelmann H D. Decision tree for optimization software. `http://plato.asu.edu/guide.html`, 2007

[18] Nelder J A, Mead R. A simplex method for function minimization. Computer Journal, 1965, 7:308–313

[19] Willems J C. Least squares stationary optimal control and the algebraic Riccati equation. IEEE Transactions on Automatic Control, 1971, 16(6):621–634

[20] The MathWorks Inc. Robust control toolbox user's manual, 2007

[21] Löfberg J. YALMIP: a toolbox for modeling and optimization in MATLAB. Proceedings of IEEE International Symposium on Computer Aided Control Systems Design. Taipei, 2004, 284–289

[22] Chipperfield A, Fleming P. Genetic algorithm toolbox user's guide. Department of Automatic Control and Systems Engineering, University of Sheffield, 1994

[23] Ackley D H. A connectionist machine for genetic hillclimbing. Boston, USA: Kluwer Academic Publishers, 1987

[24] Goldberg D E. Genetic algorithms in search, optimzation and machine learning. Reading, MA: Addison-Wesley, 1989

[25] Henrion D. A review of the global optimization toolbox for Maple, 2006. `http://www.laas.fr/~henrion/Papers/mapleglobopt.pdf`

[26] Press W H, Teukolsky S A, Vetterling W T, et al. Numerical recipes in C, second edition. Cambridge University Press. Free textbook at `http://www.nrbook.com/a/bookcpdf.php`, 1992

[27] Dawkins P. Differential equations. `http://tutorial.math.lamar.edu/pdf/DE/DE_Complete.pdf`, 2007

[28] Fehlberg E. Low-order classical Runge-Kutta formulas with step size control and their application to some heat transfer problems. Technical Report 315, NASA, 1969

[29] Forsythe G E, Malcolm M A, Moler C B. Computer methods for mathematical computations. Englewood Cliffs: Prentice-Hall, 1977

[30] Bogdanov A. Optimal control of a double inverted pendulum on a cart. Technical Report CSE-04-006, Department of Computer Science & Electrical Engineering, OGI School of Science & Engineering, OHSU, 2004

[31] Shampine L F, Thompson S. Solving DDEs in MATLAB. Applied Numerical Mathematics, 2001, 37(4):441–458

[32] Shampine L F, Kierzenka J, Reichelt M W. Solving boundary value problems for ordinary differential equation problems in MATLAB with bvp4c, 2000

[33] The MathWorks Inc. SimuLAB, a program for simulating dynamic systems, user's guide, 1990

[34] The MathWorks Inc. Simulink user's manual, 2007

[35] Xue D, Chen Y Q. MATLAB/Simulink based system simulation techniques. Beijing: Tsinghua University Press, 2002 (in Chinese)

[36] Moler C B. Numerical computing with MATLAB. MathWorks Inc, 2004

[37] Wikipedia. List of numerical analysis topics. http://en.wikipedia.org/wiki/List_of_numerical_analysis_topics, 2008

[38] Lancaster L, Salkauskas K. Curve and surface fitting: an introduction. London: Academic Press, 1986

[39] Xue D. Model reduction techniques and applications. Shenyang, China: Lecture Notes of Northeastern University, 1996

[40] Bosley M J, Lees F P. A survey of transfer function derivations from higher-order state-variable models. Automatica, 1972, 8:765–775

[41] The MathWorks Inc. Signal processing user's guide, 2007

[42] Grinstead C M, Snell J L. Grinstead and Snell's introduction to probability. The CHANCE Project: Open source textbook at http://math.dartmouth.edu/~prob/prob/prob.pdf, 2006

[43] StatSoft Inc. Electronic statistics textbook. Tulsa, OK: StatSoft. Electronics textbook at http://www.statsoft.com/textbook/stathome.html, 2007

[44] Landau D P, Binder K. A guide to Monte Carlo simulations in statistical physics. Cambridge University Press, 2000

[45] Conover W J. Practical nonparametric statistics. New York: Wiley, 1980

[46] Lu X. Applied statistics. Beijing: Tsinghua University Press, 1999 (in Chinese)

[47] Cody R P, Smith J K. Applied statistics and the SAS programming language. Prentice Hall, fifth edition, 2006

[48] The MathWorks Inc. Statistics toolbox user's manual, 2007

[49] Weisstein E W. Goldbach conjecture. From MathWorld — A Wolfram Web Resource. http://mathworld.wolfram.com/GoldbachConjecture.html

[50] Zadeh L A. Fuzzy sets. Information and Control, 1965, 8:338–353

[51] Hagan M T, Demuth H B, Beale M H. Neural network design. PWS Publishing Company, 1995

[52] Houck C R, Joines J A, Kay M G. A genetic algorithm for function optimization: a MATLAB implementation, 1995

[53] Kennedy J, Eberhart R. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks. Perth, Australia, 1995, 1942–1948

[54] Birge B K. PSOt, a particle swarm optimization toolbox for MATLAB. Proceedings of the 2003 IEEE Swarm Intelligence Symposium. Indianapolis, 2003, 182–186

[55] Trelea I C. The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters, 2003, 85(6):317–325

[56] Clerc M, Kennedy J. The particle swarm: explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation, 2002, 6(1):58–73

[57] Pawlak Z. Rough sets — theoretical aspects of reasoning about data. Boston, USA: Kluwer Academic Publishers, 1991

[58] Zhang X F. Research and program development of rough set data analysis system. Master's thesis, Northeastern University, 2004 (in Chinese)

[59] Hilfer R. Applications of fractional calculus in physics. Singapore: World Scientific, 2000

[60] Podlubny I. Fractional differential equations. San Diego: Academic Press, 1999

[61] Petráš I, Podlubny I, O'Leary P. Analogue realization of fractional order controllers. Fakulta BERG, TU Košice, 2002

[62] Oustaloup A, Levron F, Nanot F, et al. Frequency band complex non integer differentiator: characterization and synthesis. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 2000, 47(1):25–40

[63] Xue D, Zhao C N, Chen Y Q. A modified approximation method of fractional order system. Proceedings of IEEE Conference on Mechatronics and Automation. Luoyang, China, 2006, 1043–1048

[64] Podlubny I. The Laplace transform method for linear differential equations of the fractional order. Proceedings of the $9^{\rm th}$ International BERG Conference. Kosice, Slovak Republic (in Slovak), 1997, 119–119

[65] Podlubny I. Fractional-order systems and $\mathrm{PI}^{\lambda}\mathrm{D}^{\mu}$-controllers. IEEE Transactions on Automatic Control, 1999, 44(1):208–214

[66] The MathWorks Inc. Fuzzy logic toolbox user's manual, 2007