Chapter 1

Computer Mathematics Languages — An Overview

Mathematical problems are essential in almost all aspects of scientific and engineering research. The mathematical models should normally be established first, then, the solutions of the models under investigation can be obtained. Specific knowledge is required for the establishment of mathematical models, which needs the expertise of the researchers, and with the established models, the numerical and analytical approaches presented in this book can be used to solve the problems. In this chapter, a brief introduction to computer mathematical problems is given, and it will be illustrated through simple examples why computer mathematical languages should be learned. A concise history of the development of computer mathematical languages and mathematical tools will be introduced. Finally the framework of the book is presented. Also, an overview of the mathematics branches involved is given in this chapter.

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. Therefore, computers must be employed for solving these problems. There are basically two ways of 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.

We shall use the term "mathematical computation" throughout the book, whereas the term really means both numerical and analytical computation of mathematical problems. Normally, analytical solutions are explored first, and if there are no analytical solutions, numerical solutions are obtained.

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?

Solution *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*

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

and the result obtained is

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

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. Therefore, even though the results can be obtained manually, after hours of hard work, 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 within 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. Solve a polynomial equation

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

Solution *Applying the Lin–Bairstrow method, under double-precision scheme, the roots of the equation obtained are*

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

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. Below are the statements used in the example*

```
>> p=[1 9 135/4 135/2 1215/16 729/16 729/64]; roots(p) %  numeric
   p1=poly2sym(p); solve(p1) %  analytic solution of polynomial equations
```

Example 1.3 Do you remember how to find the determinant of an $n \times n$ matrix?

Solution *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 eventually be obtained from the algebraic sum of many determinants of* 1×1 *matrices, i.e., the scalar itself. Therefore, it can be concluded that the analytical solution 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* $n=25$*, the number of floating-point operations (flops) for the computation is* 9.679×10^{27}*, which amounts to 5580 years of computation on mainframe of 55 million billion* (5.5×10^{16}) *flops per second (the fastest mainframe in the world in 2014). Therefore, 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* $(n\gg 25)$*, 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

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

For $n=25$*, 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 is used, the analytical solution of the determinant of an* 80×80 *Hilbert matrix can be obtained within 2.5 seconds:*

$$\det(\boldsymbol{H})=\frac{1}{\underbrace{990301014669934778788676\cdots 000000000000}_{3790\ \textit{decimal digits, with some digits omitted}}}\approx 1.009794\times 10^{-3790}.$$

Below are the actual commands used in MATLAB for this problem

```
>> n=80; H=sym(hilb(n)); d=det(H) %
```
build Hilbert matrix, find determinant

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

$$y''+\mu(y^2-1)y'+y=0,$$

and when μ is large, i.e., $\mu=1000$, how to numerically solve the equation.

Solution *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. Two lines of statements in MATLAB are adequate in solving numerically such an equation*

```
>> mu=1000; f=@(t,x)[x(2); -mu*(x(1)^2-1)*x(2)-x(1)]; %
```
describe ODE
```
   [t,x]=ode15s(f,[0,3000],[-1;1]); plot(t,x) %
```
solve with graphics

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. Specific MATLAB functions or block diagram modeling tool Simulink can be used instead. Details of the methods will be given later in the book.

Example 1.5 Solve the linear programming problem given below

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

Solution *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 following statements can be used*

```
>> P.f=[-2 -1 -4 -3 -1]; P.Aineq=[0 2 1 4 2; 3 4 5 -1 -1];
   P.Bineq=[54 62]; P.lb=[0;0;3.32;0.678;2.57]; P.solver='linprog';
   P.options=optimset; x=linprog(P) % solve linear programming problem
```

and 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 $x_1 = 19, x_2 = 0, x_3 = 4, x_4 = 10, x_5 = 5$.

```
>> P.solver='intlinprog'; P.intcon=1:5; x=intlinprog(P)
```

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

Example 1.7 With the rapid development of modern science and technology, many new mathematics branches, such as fuzzy set, rough set, artificial neural network, evolutionary computing algorithms have emerged. It would be a hard and time consuming task to use the branches to solve particular problems, without using specific computer tools. If low-level programming is expected, the researcher needs to fully understand the technical contents of the branches, as well as how to implement the algorithms with computer languages. However, if the existing tools and frameworks are used instead, the problems can be solved in a much simpler manner.

In many subjects, such as electric circuits, electronics, motor drive, power electronics, 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 $\frac{2}{\sqrt{\pi}} \int_0^a \mathrm{e}^{-x^2}\,\mathrm{d}x$ 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 (429–500), 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 practices. Even with the imprecise value 3.14 found by Archimedes (B.C. 287–B.C. 212) 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 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 ($\boldsymbol{W}_{\mathrm{R}}$, $\boldsymbol{W}_{\mathrm{I}}$ 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.

1.1.4 Limitations of conventional computer languages

Many people are using conventional computer languages, such as C and Fortran, in their research. Needless to say that these languages were very useful, and they were the low-level supporting languages of the computer mathematics languages such as MATLAB. However, for the modern scientific and engineering researchers, these languages are not adequate for solving their complicated computational problems. For instance, even for very experienced C programmers, they may not be able to write C code to find the indefinite integral of a given function, these involve knowledge of mathematical mechanization. Even for numerical computations, there are limitations. Here two examples are given to illustrate the problems.

Example 1.8 It is known that Fibonacci sequence can be generated with the following recursive formula, $a_1 = a_2 = 1$, and $a_k = a_{k-1} + a_{k-2}$, $k = 3, 4, \cdots$. Please compute its first 100 terms.

Solution *Data type for each variable is assigned in C programming language first. Since*

the terms in the sequence are integers, it is natural to select the data types `int` *or* `long`*. If* `int` *is selected, the following C program can be written.*

```
main()
{  int a1, a2, a3, i;
   a1=1; a2=1; printf("%d  %d  ",a1,a2);
   for (i=3; i<=100; i++){a3=a1+a2; printf("%d  ",a3); a1=a2; a2=a3;
}}
```

It seems that the sequence can be obtained simply by using the program. But wait. Are the results obtained correct? If the program is executed, from the 24th term, the value of the sequence is negative, and from that term on, the terms are sometimes positive, sometimes negative. It is obvious that some peculiar things must have happened in the program. The problem is caused by the `int` *data type, since the range of it is* $(-32767, 32767)$*. If a term is beyond this range, wrong results are generated. Even if* `long` *data type is adopted instead, the correct answers may only last till about 10 more terms. To solve the problem of finding the first 100 terms, or even more, of Fibonacci sequence is certainly beyond the capabilities of average C users, or even experienced C programmers. Extremely slight carelessness may lead to misleading results.*

With the use of MATLAB, this kind of trivial thing should not be considered. The following code can be written directly

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

Besides, for more accurate representation of the terms, symbolic data type can be used instead, by substituting the first statement with $\boldsymbol{a} =$ `sym([1,1])`*. In this case, the 100th term is* $a_{100} = 354224848179261915075$*, and even more, the 10000th term may be obtained, with about 32 seconds of computation, and all the 2089 decimal digits can be found, whose display may occupy more than half a page of the book.*

Example 1.9 Write a universal C program to compute the product of two matrices.

Solution *It is known from linear algebra courses that if matrix* $\boldsymbol{A}$ *is an* $n \times p$ *one, while* $\boldsymbol{B}$ *is a* $p \times m$ *one, the product of the two matrices can be obtained with*

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj},\ i = 1, \cdots, n,\ \ j = 1, \cdots, m$$

From the formula, the kernel part of the program is the triple loop structure

```
for (i=0; i<n; i++){for (j=0; j<m; j++){
   c[i][j]=0; for (k=0; k<p; k++) c[i][j]+=a[i][k]*b[k][j]; }}
```

It seems again that the problem can be solved with these simple statements. Unfortunately there is still a serious problem in the short code, the multiplicability of the two matrices is not considered. Imprecisely speaking, when the number of columns of $\boldsymbol{A}$ *equals the number of rows of* $\boldsymbol{B}$*, the product can be found, otherwise, they are not multiplicable. To solve the problem, an extra* `if` *statement should be added*

`if` *cols_of_*$\boldsymbol{A}$ `!=` *rows_of_*$\boldsymbol{B}$`,` *display an error message*

Unfortunately by introducing such a statement, a new problem emerges. In mathematics, when $\boldsymbol{A}$ *or* $\boldsymbol{B}$ *is a scalar, the product of* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *can be found, however, this case is expelled by introducing the above* `if` *statement. To solve the problem, more* `if` *statements are expected to check the scalar cases.*

Although the above modifications are made, this program is not a universal one, since complex matrices were not considered ar all. More statements are needed to make the program universal.

It can be seen from the example that, if C or similar computer languages are used, the programmers must be very careful to consider all the possible cases. If one or more cases were not considered, wrong or misleading results may be obtained. In MATLAB, this kind of trivial thing need not to be considered at all. The command $\boldsymbol{C} = \boldsymbol{A}*\boldsymbol{B}$ *can be used directly. If the two matrices are multiplicable, the product can be obtained, otherwise, an error message will be displayed to indicate why the product cannot be found.*

Of course, in real-time control and similar areas, C or similar languages have their own advantages. Although some of the MATLAB code can be translated into C automatically, this is beyond the scope of this book.

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 then most advanced EISPACK and LINPACK packages were too complicated. MATLAB (MATrix LABoratory) was then conceived and developed. The first release of MATLAB was freely distributed in late 1970's. Cleve Moler and Jack Little co-founded The MathWorks Inc. in 1984 to develop the MATLAB language [5]. 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 a numerical mathematician, 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 MathWorks Inc., 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 was used to solve symbolic computation problems, and now with the MuPAD engine. Therefore, the symbolic computation capability of MATLAB is essentially as good as Mathematica and

Maple for most engineering 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 codes are not accessible to the users. Therefore, 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 to 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 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.

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 to 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.

1.3.1 The organization of the book

The contents of the book are summarized below:

Chapter 1, the current chapter, answers the question "Why MATLAB" in scientific computation and 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, three-dimensional graphics, or even four-dimensional graphics, through volume visualization techniques, 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 univariate and multivariate functions. Series expansion problems such as Taylor series and Fourier series expansions as well as series sums and products are covered. Convergency tests of infinite series are explored. MATLAB solutions to path, line and surface integrals are illustrated. Finally, numerical differentiation and integral (or quadrature) are also introduced.

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. Also, the integer power of matrices are also explored.

Chapter 5, "Integral Transforms and Complex-valued Functions," includes the solutions to Laplace transform problems and their inverse, Fourier transforms and their variations, z, Mellin and Hankel transforms. Numerical solutions of integral transform are illustrated, where numerical inverse Laplace transform and fast Fourier transform are in particular discussed. The analysis of complex-valued functions are also introduced, including poles, residues, partial fraction expansion, Laurent series, and closed-path integral problems, all with many illustrative solution examples. Difference equation solutions are explored also in the chapter.

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. Multi-objective optimization problems and dynamical programming problems are also introduced. Dynamic programming problems, with particular applications in shortest path planning problems, are demonstrated.

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. In particular, block diagram-based modeling and numerical solutions of differential equations are explored with the sophisticated Simulink environment.

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. An introduction to signal filtering and de-noising problems is also presented briefly in this chapter.

Chapter 9, "Probability and Mathematical Statistics Problems," studies the probability distributions and pseudorandom 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 demonstrated briefly through examples. An introduction to principal component analysis problems is also given.

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, evolutionary optimization methods including genetic algorithms and particle swarm optimization methods. Most interestingly, fractional-order calculus (derivatives or integrals of non-integer order) problems are introduced with basic numerical computational techniques and examples.

1.3.2 How to learn and use MATLAB?

The best way to learn a computer language like MATLAB is learning through extensive practice. MATLAB should be installed on your computer, and when invoked, the main interface is shown in Figure 1.1. The Command Window is the place where MATLAB commands are issued, and >> is the MATLAB prompt. For new users of MATLAB, it is advised to type `demo` after the MATLAB prompt, and experience the facilities provided in an easy-to-understand manner.

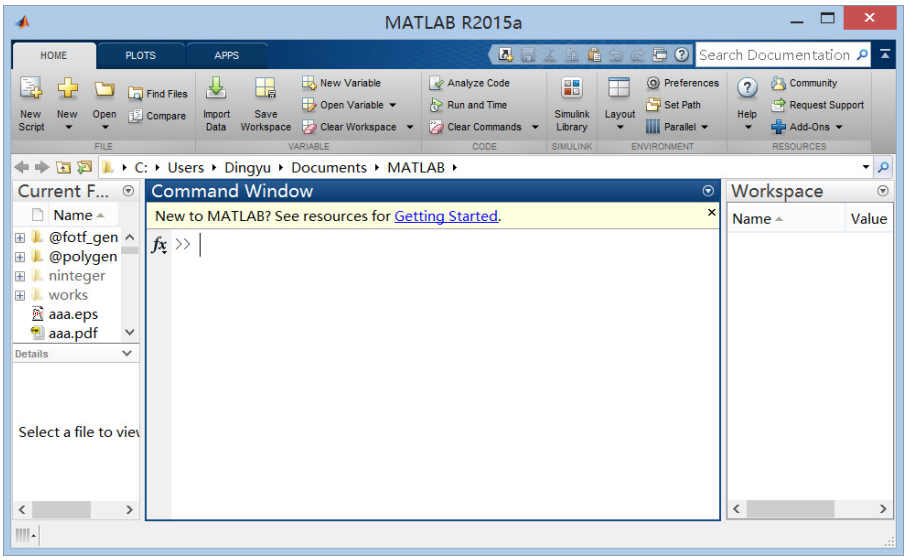


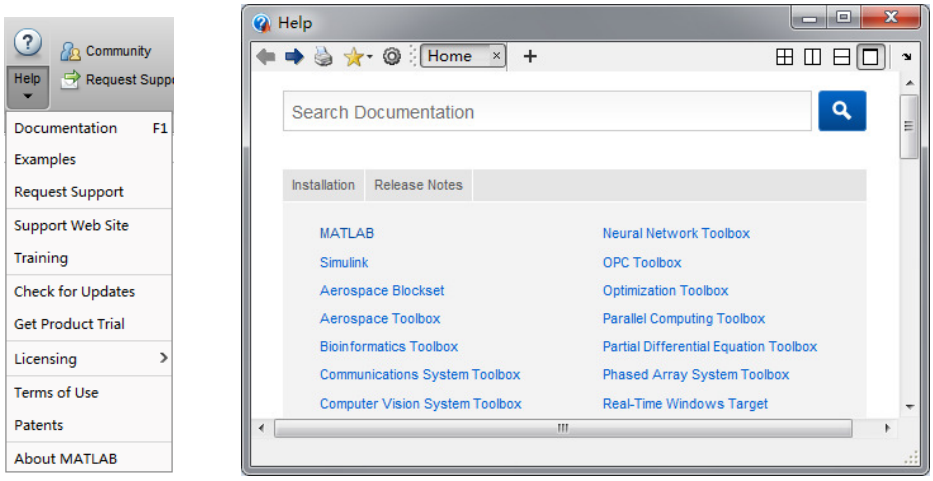
FIGURE 1.1: Main interface of MATLAB.

In this book, almost all the engineering mathematics branches are involved. On-line help facilities provided in MATLAB are useful for getting extra information of a particular function or a class of problems. The readers are recommended to get acquainted with such facilities. Help information can be obtained with the Help menu in the MATLAB command window, as shown in Figure 1.2(a). If the Using the Desktop menu item is selected, an on-line help window is opened, as shown in Figure 1.2(b).

Alternatively, the commands `help` or `doc` can be issued in the MATLAB command window, and the command `lookfor` can be used to search keywords.

1.3.3 The three-phase solution methodology

A *three-phase methodology* proposed by the authors is used throughout the book in presenting mathematical problem solutions [6]. The three phases are respectively "What," "How" and "Solve." In the "What" phase, the physical explanation of the mathematical problem to be solved is presented. Even though the students have not yet learned the



(a) the Help menu (b) on-line help window

FIGURE 1.2: On-line help information.

corresponding mathematics course, he can understand roughly what the problem is really about. In the "How" phase, the mathematical problem should be described in a manner understandable by MATLAB. In the final "Solve" phase, appropriate MATLAB functions should be called to solve the problem directly. If there is an existing MATLAB function, the syntax of the function is presented, and if there is not one, a universal function is written to solve the problem.

Example 1.10 Let us revisit the materials in Example 1.5, where the linear programming problem is involved, the philosophy used in the book is illustrated

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

Solution *To solve such a problem, even though the reader may have not learned optimization courses, the solution can be obtained following the three-phases.*

(i) ***"What" phase*** *In this book, we shall explain first the physical meaning of the mathematical problem. In this particular example, the mathematical formula means that under the simultaneous inequality constraints*

$$\begin{cases} 2x_2 + x_3 + 4x_4 + 2x_5 \leqslant 54 \\ 3x_1 + 4x_2 + 5x_3 - x_4 - x_5 \leqslant 62 \\ x_1, x_2 \geqslant 0,\ x_3 \geqslant 3.32,\ x_4 \geqslant 0.678,\ x_5 \geqslant 2.57, \end{cases}$$

how can we find a set of decision variables x_i, to minimize the objective function

$$f(\boldsymbol{x}) = -2x_1 - x_2 - 4x_3 - 3x_4 - x_5.$$

(ii) ***"How" phase*** *Illustrate the readers how to represent the problem in MATLAB. The code in Example 1.5 can be used to establish variable* `P`*. The mathematical problem should be expressed in a format understandable by MATLAB.*

```
>> P.f=[-2 -1 -4 -3 -1]; P.Aineq=[0 2 1 4 2; 3 4 5 -1 -1];
```

`P.Bineq=[54 62]; P.lb=[0;0;3.32;0.678;2.57]; P.solver='linprog';`
`P.options=optimset;` `%` express the linear programming problem in structure `P`

(iii) ***"Solve" phase*** *Call the* `linprog()` *function and get the result.*

`>> x=linprog(P)` `%` solve the problem with appropriate function `linprog`

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 bypass 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' capability in tackling mathematical problems can be significantly enhanced after reading this book.

Exercises

Exercise 1.1 *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.*

Exercise 1.2 *In order to understand the three-phase learning stage, please revisit Example 1.5, and see which statements belong to the second and third phases. Also, please think about what should be done for the first phase.*

Exercise 1.3 *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.*

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

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, Molor 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

[5] Moler C B. Evolution of MATLAB (Video of presentation at Tongji University, China, subtitled by Xue D Y). `http://v.youku.com/v_show/id_XNDc0NTM4NzQw.html?tpa=dW5pb25faWQ9MjAwMDE0XzEwMDAwMV8wMV8wMQ`, 2012

[6] Xue D Y. Mathematics education made more practical with MATLAB. Presentation at the First MathWorks Asian Research Faculty Summit, Tokyo, Japan, November, 2014