

MATLAB[®] for Engineers

Holly Moore

Sixth Edition



MATLAB[®] for Engineers

This page intentionally left blank

MATLAB[®] for Engineers

Sixth Edition

HOLLY MOORE

Salt Lake Community College
Salt Lake City, Utah



Content Management: Dawn Murrin, Erin Ault
Content Production: Devanshi Tyagi
Product Management: Holly Stark
Product Marketing: Krista Clark
Rights and Permissions: Rimpay Sharma

Please contact <https://support.pearson.com/getsupport/s/> with any queries on this content.

Microsoft and/or its respective suppliers make no representations about the suitability of the information contained in the documents and related graphics published as part of the services for any purpose. All such documents and related graphics are provided “as is” without warranty of any kind. Microsoft and/or its respective suppliers hereby disclaim all warranties and conditions with regard to this information, including all warranties and conditions of merchantability, whether express, implied or statutory, fitness for a particular purpose, title and non-infringement. In no event shall Microsoft and/or its respective suppliers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of information available from the services.

The documents and related graphics contained herein could include technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Microsoft and/or its respective suppliers may make improvements and/or changes in the product(s) and/or the program(s) described herein at any time. Partial screen shots may be viewed in full within the software version specified.

Microsoft and Windows are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Copyright © 2023, 2018, 2015 by Pearson Education, Inc. or its affiliates, 221 River Street, Hoboken, NJ 07030. All Rights Reserved. Manufactured in the United States of America. This publication is protected by copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise. For information regarding permissions, request forms, and the appropriate contacts within the Pearson Education Global Rights and Permissions department, please visit www.pearsoned.com/permissions/.

Acknowledgments of third-party content appear on the appropriate page within the text.

PEARSON is an exclusive trademark owned by Pearson Education, Inc. or its affiliates in the U.S. and/or other countries.

Unless otherwise indicated herein, any third-party trademarks, logos, or icons that may appear in this work are the property of their respective owners, and any references to third-party trademarks, logos, icons, or other trade dress are for demonstrative or descriptive purposes only. Such references are not intended to imply any sponsorship, endorsement, authorization, or promotion of Pearson's products by the owners of such marks, or any relationship between the owner and Pearson Education, Inc., or its affiliates, authors, licensees, or distributors.

MATLAB® and Simulink® are registered trademarks of The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA 01760-2098.

Library of Congress Cataloging-in-Publication Data

Names: Moore, Holly, author.

Title: MATLAB® for Engineers / Holly Moore, Salt Lake Community College.

Description: Sixth edition. | Hoboken : Pearson, [2023] | Includes index.

Identifiers: LCCN 2021051256 | ISBN 9780137627981

Subjects: LCSH: Engineering mathematics—Data processing. | MATLAB.

Classification: LCC TA345 .M585 2022 | DDC 620.001/51—dc23/eng/20211117

LC record available at <https://lcn.loc.gov/2021051256>

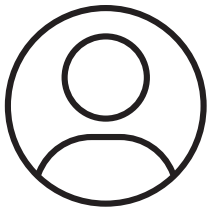
ScoutAutomatedPrintCode



Rental:

ISBN-10: 0-13-762798-X

ISBN-13: 978-0-13-762798-1



Pearson's Commitment to Diversity, Equity, and Inclusion

Pearson is dedicated to creating bias-free content that reflects the diversity, depth, and breadth of all learners' lived experiences.

We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, sex, sexual orientation, socioeconomic status, ability, age, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where:

- Everyone has an equitable and lifelong opportunity to succeed through learning.
- Our educational content accurately reflects the histories and lived experiences of the learners we serve.
- Our educational products and services are inclusive and represent the rich diversity of learners.
- Our educational content prompts deeper discussions with students and motivates them to expand their own learning (and worldview).

Accessibility

We are also committed to providing products that are fully accessible to all learners. As per Pearson's guidelines for accessible educational Web media, we test and retest the capabilities of our products against the highest standards for every release, following the WCAG guidelines in developing new products for copyright year 2022 and beyond.



You can learn more about Pearson's commitment to accessibility at

<https://www.pearson.com/us/accessibility.html>

Contact Us

While we work hard to present unbiased, fully accessible content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.



Please contact us with concerns about any potential bias at
<https://www.pearson.com/report-bias.html>



For accessibility-related issues, such as using assistive technology with Pearson products, alternative text requests, or accessibility documentation, email the Pearson Disability Support team at disability.support@pearson.com



This page intentionally left blank

Contents

ABOUT THIS BOOK	xiii
DEDICATION AND ACKNOWLEDGMENTS	xvii

1 • ABOUT MATLAB	1
-------------------------	----------

1.1 What Is MATLAB?	1
1.2 Student Edition of MATLAB	2
1.3 How is MATLAB Used in Industry?	3
1.4 Problem Solving in Engineering and Science	5

2 • MATLAB ENVIRONMENT	9
-------------------------------	----------

2.1 Getting Started	9
2.2 MATLAB Windows	11
2.3 Solving Problems with MATLAB	17
2.4 Saving Your Work	41
Summary	53
Key Terms	56
Problems	56

3 • BUILT-IN MATLAB FUNCTIONS	64
--------------------------------------	-----------

Introduction	64
3.1 Using Built-In Functions	64
3.2 Using the Help Feature	66
3.3 Elementary Math Functions	67
3.4 Trigonometric Functions	75
3.5 Data Analysis Functions	80
3.6 Random Numbers	100
3.7 Complex Numbers	104
3.8 Computational Limitations	108
3.9 Special Values and Miscellaneous Functions	109
Summary	111
Key Terms	113
Problems	114

4 • MANIPULATING MATLAB ARRAYS 121

Introduction	121
4.1 Manipulating Numeric Arrays	121
4.2 Problems with Two Variables—Using Meshgrid	128
4.3 Special Arrays	135
4.4 Introduction to Character and String Arrays	141
Summary	146
Key Terms	147
Problems	148

5 • PLOTTING 156

Introduction	156
5.1 Two-Dimensional Plots	156
5.2 Subplots—Tiled Chart Layouts	173
5.3 Other Types of Two-Dimensional Plots	176
5.4 Three-Dimensional Plotting	193
5.5 Editing Plots from the Menu Bar	200
5.6 Creating Plots from the Workspace Window	202
5.7 Saving Your Plots	203
5.8 Other Plotting Options	203
Summary	204
Key Terms	206
Problems	207

6 • USER-DEFINED FUNCTIONS 218

Introduction	218
6.1 Creating Function Files	218
6.2 Subfunctions	238
6.3 Creating Your Own Toolbox of Functions	243
6.4 Anonymous Functions and Function Handles	245
6.5 Function Functions	246
Summary	247
Key Terms	249
Problems	249

7 • USER-CONTROLLED INPUT AND OUTPUT 257

Introduction	257
7.1 User-Defined Input	257
7.2 Output Options	262
7.3 Graphical Input	276
7.4 Reading and Writing Data from Files	277
7.5 Debugging Your Code	280
Summary	285
Key Terms	286
Problems	287

8 • LOGICAL FUNCTIONS AND SELECTION STRUCTURES 291

Introduction	291
8.1 Relational and Logical Operators	292
8.2 Flowcharts and Pseudocode	294
8.3 Logical Functions	296
8.4 Logical Indexing	303
8.5 Selection Structures	307
8.6 Debugging	324
Summary	325
Key Terms	326
Problems	326

9 • REPETITION STRUCTURES 341

Introduction	341
9.1 <code>for</code> Loops	342
9.2 <code>while</code> Loops	350
9.3 <code>break</code> and <code>continue</code>	358
9.4 Midpoint Break Loops	359
9.5 Nested Loops	363
9.6 Improving the Efficiency of Loops	364
Summary	367
Key Terms	368
Problems	369

10 • MATRIX ALGEBRA 374

Introduction	374
10.1 Matrix Operations and Functions	374
10.2 Solutions of Systems of Linear Equations	396
10.3 Special Matrices	409
Summary	410
Key Terms	412
Problems	412

11 • OTHER KINDS OF ARRAYS 420

Introduction	420
11.1 Numeric Data Types	421
11.2 Character and String Data	427
11.3 Symbolic Data	432
11.4 Logical Data	432
11.5 Sparse Arrays	434
11.6 Categorical Arrays	434
11.7 Time Arrays	435
11.8 Multidimensional Arrays	439
11.9 Cell Arrays	441
11.10 Structure Arrays	443

11.11 Table Arrays	449
11.12 Timetable Arrays	458
Summary	462
Key Terms	464
Problems	464

12 • SYMBOLIC MATHEMATICS

472

Introduction	472
12.1 Symbolic Algebra	472
12.2 Solving Expressions and Equations	480
12.3 Symbolic Plotting	490
12.4 Units of Measurement	498
12.5 Calculus	503
12.6 Differential Equations	516
12.7 Converting Symbolic Expressions to Anonymous Functions	519
Summary	520
Key Terms	522
Problems	523

13 • NUMERICAL TECHNIQUES

532

13.1 Interpolation	532
13.2 Curve Fitting	542
13.3 Using the Interactive Fitting Tools	554
13.4 Differences and Numerical Differentiation	557
13.5 Numerical Integration	565
13.6 Solving Differential Equations Numerically	570
Summary	578
Key Terms	581
Problems	581

14 • ADVANCED GRAPHICS

589

Introduction	589
14.1 Images	589
14.2 Graphics Objects	604
14.3 Animation	611
14.4 Other Visualization Techniques	618
14.5 Introduction to Volume Visualization	620
Summary	625
Key Terms	627
Problems	627

15 • SIMULINK®—A BRIEF INTRODUCTION

630

Introduction	630
15.1 Applications	630
15.2 Getting Started	631
15.3 Solving Differential Equations with Simulink	639
Summary	646
Key Terms	646
Problems	646

APPENDIX A • SPECIAL CHARACTERS, COMMANDS, AND FUNCTIONS

651

APPENDIX B • SCALING TECHNIQUES

666

APPENDIX C • ANNUAL CLIMATOLOGICAL SUMMARY

669

APPENDIX D • SOLUTIONS TO PRACTICE EXERCISES

(Available at www.pearsonhighered.com/moore)

INDEX

671

This page intentionally left blank

About This Book

This book grew out of my experience teaching MATLAB and other computing languages to freshmen engineering students at Salt Lake Community College. I was frustrated by the lack of a text that “started at the beginning.” Although there were many comprehensive reference books, they assumed a level of both mathematical and computer sophistication that my students did not possess. Also, because MATLAB was originally adopted by practitioners in the fields of signal processing and electrical engineering, most of these texts provided examples primarily from those areas, an approach that didn’t fit with a general engineering curriculum. This text starts with basic algebra and shows how MATLAB can be used to solve engineering problems from a wide range of disciplines. The examples are drawn from concepts introduced in early chemistry and physics classes and freshman and sophomore engineering classes. A standard problem-solving methodology is used consistently.

The text assumes that the student has a basic understanding of college algebra and has been introduced to trigonometric concepts; students who are mathematically more advanced generally progress through the material more rapidly. Although the text is not intended to teach subjects such as statistics or matrix algebra, when the MATLAB techniques related to these subjects are introduced, a brief background is included. In addition, sections describing MATLAB techniques for solving problems by means of calculus and differential equations are introduced near the end of appropriate chapters. These sections can be assigned for additional study to students with a more advanced mathematics background, or they may be useful as reference material as students progress through an engineering curriculum.

The book is intended to be a “hands-on” manual. My students have been most successful when they read the book while sitting beside a computer and typing in the examples as they go. Numerous examples are embedded in the text, with more complicated numbered examples included in each chapter to reinforce the concepts introduced. Practice exercises are included in each chapter to give students an immediate opportunity to use their new skills.

The material is grouped into three sections. The first, *An Introduction to Basic MATLAB Skills*, gets the student started and contains the following chapters:

- Chapter 1 shows how MATLAB is used in engineering and introduces a standard problem-solving methodology.
- Chapter 2 introduces the MATLAB environment and the skills required to perform basic computations. It also introduces MATLAB program files and the concept of organizing code into sections. Doing so early in the text makes it easier for students to save their work and develop a consistent programming strategy.
- Chapter 3 details the wide variety of problems that can be solved with built-in MATLAB functions. Background material on many of the functions is provided to help the student understand how they might be used. For example, the difference between Gaussian random numbers and uniform random numbers is described, and examples of each are presented.

- Chapter 4 demonstrates the power of formulating problems by using arrays in MATLAB and expanding on the techniques employed to define those arrays. The `meshgrid` function is introduced in this chapter and is used to solve problems with two variables. The difficult concept of meshing variables is revisited in Chapter 5 when surface plots are introduced. Chapter 4 also introduces how arrays are used for storing and manipulating text, and how the same techniques employed to manipulate numeric arrays apply to text arrays as well.
- Chapter 5 describes the wide variety of both two-dimensional and three-dimensional plotting techniques available in MATLAB. Creating plots via MATLAB commands, either from the command window or from within a MATLAB program, is emphasized. The extremely valuable techniques of interactively editing plots and creating plots directly from the workspace window are also introduced.

MATLAB is a powerful programming language that includes the basic constructs common to most programming languages. Because it is a scripting language, creating programs and debugging them in MATLAB is often easier than in traditional programming languages such as C++. This makes MATLAB a valuable tool for introductory programming classes. The second section of the text, *Programming in MATLAB*, introduces students to programming and consists of the following chapters:

- Chapter 6 describes how to create and use user-defined functions. It also teaches students how to create a “toolbox” of functions to use in their own programming projects.
- Chapter 7 introduces functions that interact with the program user, including user-defined input, formatted output, and graphical input techniques. The use of MATLAB’s debugging tools is also introduced.
- Chapter 8 describes logical functions such as `find`, the use of logical indexing, and demonstrates how these approaches vary from the `if` and `if/else` structures. The `switch/case` structure is also introduced. The use of logical functions and logical indexing over control structures is emphasized, partly because students (and teachers) who have previous programming experience often overlook the advantages of using MATLAB’s built-in matrix functionality.
- Chapter 9 introduces repetition structures, including `for` loops, `while` loops, and midpoint break loops that utilize the `break` command. Numerous examples are included because students find these concepts particularly challenging.

Chapters 1 through 9 should be taught sequentially, but the chapters in Section 3, *Advanced MATLAB Concepts*, do not depend upon each other. Any or all of these chapters could be used in an introductory course or could serve as reference material for self-study. Most of the material is appropriate for freshmen. A two-credit course might include Chapters 1 through 9 plus Chapter 10, while a three-credit course might include Chapters 1 through 14, but eliminate Sections 12.5, 12.6, 13.4, 13.5, and 13.6, which describe differentiation techniques, integration techniques, and solution techniques for differential equations. The skills developed in these chapters will be especially useful as students become more involved in solving engineering problems:

- Chapter 10 discusses problem solving with matrix algebra, including dot products, cross products, and the solution of linear systems of equations. Although matrix algebra is widely used in all engineering fields, it finds early application in the statics and dynamics classes taken by most engineering majors.

- Chapter 11 is an introduction to the wide variety of data types available in MATLAB.
- Chapter 12 introduces MATLAB's symbolic mathematics package. My students tell me that the package is one of the most valuable sets of techniques introduced in the course. It is something they start using immediately.
- Chapter 13 presents numerical techniques used in a wide variety of applications, especially curve fitting and statistics. Students value these techniques when they take laboratory classes such as chemistry or physics or when they take the labs associated with engineering classes such as heat transfer, fluid dynamics, or strengths of materials.
- Chapter 14 examines graphical techniques used to visualize data. These techniques are especially useful for analyzing the results of numerical analysis calculations, including results from structural analysis, fluid dynamics, and heat transfer codes.
- Chapter 15 introduces Simulink, which is a simulation package built on top of the MATLAB platform. Simulink uses a graphical user interface that allows programmers to build models of dynamic systems. It has found significant acceptance in the field of electrical engineering but has wide application across the engineering spectrum.

Appendix A lists all of the functions and special symbols (or characters) introduced in the text. Appendix B describes strategies for scaling data, so that the resulting plots are linear. Appendix C includes the Asheville, North Carolina, weather data used in a number of the example problems.

All instructor resources are available for download at www.pearson.com. If you are in need of a login and password for this site, please contact your local Pearson representative. The instructor resources include the following material:

- Solutions to practice exercises (These files are also available on the student version of the website.)
- Solutions to example problems
- Solutions to homework problems
- PowerPoint slides for each chapter
- All of the figures used in the text, suitable for inclusion in your own PowerPoint presentations
- A series of lectures (including narration) suitable for use with online classes or as reviews

Appendix D Solutions to Practice Exercises can be found at the following website:
www.pearsonhighered.com/moore

WHAT'S NEW IN THIS EDITION

New versions of MATLAB are rolled out every six months, which makes keeping any text up-to-date a challenge. The changes in this edition reflect software updates up through R2021b, which include the following:

- Screen shots shown in the book are updated to reflect the 2021b release.
- The use of subfunctions (also called local functions) in MATLAB programs is updated.
- The use of Live Scripts is emphasized. Both Scripts (M-files) and Live Scripts (MLX-files) are used to illustrate how to perform tasks. All of the examples,

practice exercises, and homework problems are available as both scripts and live scripts on the Pearson website at www.pearsonhighered.com/moore.

- The use of the string data type instead of character arrays is emphasized. The benefits of this change in programming strategy is discussed. Cell arrays of character arrays are still taught, so that students will recognize and understand the program structures when they see it in ‘legacy’ code.
- Tables are emphasized over cell and structure arrays as the primary storage medium for data sets containing multiple data types.
- A number of new functions are introduced throughout the book.
- Additional problems are added and some problems are modified, based on the feedback from both instructors and students who have used the book. Historic data used in the problems has been updated to current values—for example, ACE hurricane information now includes data through 2021.

Dedication and Acknowledgments

This project would not have been possible without the support of both my family and colleagues. Thanks to Mike, Heidi, Meagan, and David, and to my husband, Dr. Steven Purcell. I also benefited greatly from the suggestions for problems related to electricity from Lee Brinton and Gene Riggs of the SLCC Electrical Engineering Department. Their cheerful efforts to educate me on the mysteries of electricity are much appreciated. I'd also like to thank Quentin McRae, also at SLCC, who made numerous suggestions that improved the homework problems. And finally, Art Fox has been my tireless colleague and collaborator for almost 20 years and is responsible in large part for the success of our MATLAB computing courses at SLCC—especially the online versions.

This book is dedicated to my father, Professor George E. Moore, who taught in the Department of Electrical Engineering at the South Dakota School of Mines and Technology for almost 20 years. Professor Moore earned his college degree at the age of 54 after a successful career as a pilot in the United States Air Force and was a living reminder that you are never too old to learn. My mother, Jean Moore, encouraged both him and her two daughters to explore outside the box. Her loving support made it possible for both my sister and I to enjoy careers in engineering—something few women attempted in the early 1970s. I hope that readers of this text will take a minute to thank those people in their lives who've helped them make their dreams come true. Thanks Mom and Dad!

This page intentionally left blank



CHAPTER

1

About MATLAB

Objectives

After reading this chapter, you should be able to:

- Describe what MATLAB is and why it is widely used in engineering and science.
- Describe the advantages and limitations of the student edition of MATLAB.
- Identify example applications where MATLAB is used in industry to solve problems.
- Formulate problems using a structured problem-solving approach.

1.1 WHAT IS MATLAB?

KEY IDEA

MATLAB is a multiparadigm programming language.

MATLAB is a “**multiparadigm**” programming language, which means it can be used in several different ways to perform programming tasks. Those tasks can range from something as simple as balancing your checkbook to complex engineering modeling applications. The way you approach the calculations required to complete these tasks may be different, depending on your experience and personal preferences, the number of calculations required, the amount of data involved, mathematical limitations, the audience for the results, and many other considerations.

MATLAB is a versatile language, but it can’t do everything. It is a large **application program** designed to optimize mathematical calculations; number-crunching is where it shines. You wouldn’t want to use it to write a **word-processing program**, for example. If you wanted to write an **operating system** or design software, C++, Java, or some other higher-level language would be a better choice. In fact, MATLAB was originally written in Fortran and then later rewritten in C. So why use MATLAB instead of the underlying **high-level language** for any programming task? It is easier to formulate mathematical problems in MATLAB, the use of array calculations makes repetitive calculations run faster in MATLAB than in the underlying language, and there is easy access to graphics to help you display your results.

There are several different ways to access MATLAB, which is produced by the MathWorks. It is available in professional, student, and home versions. The professional version is probably installed in your college or university computer laboratory,

KEY IDEA

MATLAB is optimized for array calculations.

KEY IDEA

MATLAB offers easy access to graphics.

but you may be able to download a copy if your institution has a **site license**. It is also available online if your institution has a license or if you have purchased either a student license or a home license. With the online version, you can use MATLAB in a **web browser** without installing any software on your personal computer. MATLAB is updated regularly; this textbook is based on MATLAB 9.10. If you are using a different version, you may notice a difference in the layout of the **graphical user interface**; however, the differences in the coding approaches should be minor.

KEY IDEA

MATLAB toolboxes offer additional capability and are purchased separately.

The standard installation of MATLAB can solve a wide variety of technical problems and is sufficient for many applications. However, additional capability is available in the form of **function** toolboxes. These **toolboxes** are purchased separately. If you have access to the complete suite of toolboxes, you may be tempted to download everything—however, the complete R2021b version of the software required almost 28 gigabytes of memory. Future versions are sure to be larger. It would be wise to carefully consider your computer's memory storage limits and your internet download speed before deciding which toolboxes to download. You can find a complete list of the MATLAB product family at the MathWorks website, www.mathworks.com.

1.2 STUDENT EDITION OF MATLAB

The professional and student editions of MATLAB are very similar. Beginning students probably won't be able to tell the difference. Student editions are available for Microsoft Windows, Mac, and Linux operating systems, and can be purchased from college bookstores or online from MathWorks at www.mathworks.com.

KEY IDEA

MATLAB is regularly updated.

MathWorks packages its software in groups called *releases*, and MATLAB 9.10 is featured, along with other products, such as Simulink, in Release R2021b. New versions are released every six months. Students may purchase just MATLAB, or a bundle that includes the following products:

- A single-user license, limited to students for use in their classwork (The professional version is licensed either singly or to a group.)
- Control Systems Toolbox
- Curve Fitting Toolbox
- DSP System Toolbox
- Full MATLAB
- Image Processing Toolbox
- Instrument Control Toolbox
- Optimization Toolbox
- Parallel Computing Toolbox
- Signal Processing Toolbox
- Simulink
- Statistics and Machine Learning Toolbox
- Symbolic Math Toolbox

Toolboxes other than those included with the student edition may be purchased separately. If you are using a professional installation of MATLAB, all of the toolboxes available in the student edition may not be available to you.

The biggest difference you should notice between the professional and student editions is the command prompt, which is

`>>`

in the professional version, and

`EDU>>`

in the student edition.

1.3 HOW IS MATLAB USED IN INDUSTRY?

The ability to use tools such as MATLAB is quickly becoming a requirement for many engineering positions. A recent job search on Monster.com found the following advertisement:

. . . is looking for a System Test Engineer with **Avionics** experience. Responsibilities include modification of MATLAB **scripts**, execution of Simulink simulations, and analysis of the results data. Candidate **MUST** be very familiar with MATLAB, Simulink, and C++...

This advertisement isn't unusual. The same search turned up 771 different companies that specifically required MATLAB skills for entry-level engineers. Widely used in engineering and science fields, MATLAB is particularly popular for electrical engineering applications. The sections that follow outline a few of the many applications currently using MATLAB.

KEY IDEA

MATLAB is widely used in engineering.

1.3.1 Electrical Engineering

MATLAB is used in electrical engineering for a wide variety of applications. For example, Figure 1.1 includes several images created to help visualize the arrangements of **electromagnetic fields** in space and time. These images represent real physical situations with practical applications. Cellular communications, medical diagnostics, and home computers are just a few of the technologies that exist thanks to our understanding of this beautiful phenomenon.

1.3.2 Biomedical Engineering

Medical images are usually saved as dicom files (the Digital Imaging and Communications in Medicine standard). Dicom files use the file extension .dcm. The MathWorks offers an Image Processing Toolbox that can read these files, making their data available to MATLAB. (The Image Processing Toolbox is included with the student edition, and is optional with the professional edition.) The Image Processing Toolbox also includes a wide range of functions, many of them especially appropriate

Figure 1.1

Arrangements of electromagnetic fields.
(a) **surface plasmon** polariton; (b) light scattering by a circular metal cylinder; (c) beam forming by a six-element dipole array.
(Used with permission of Dr. James R. Nagel, University of Utah Department of Electrical and Computer Engineering.)

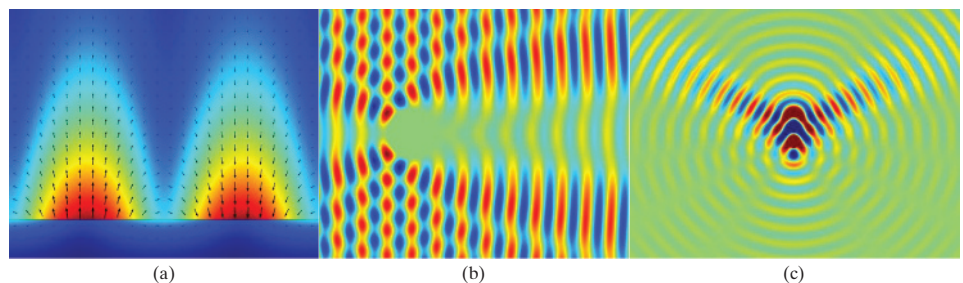
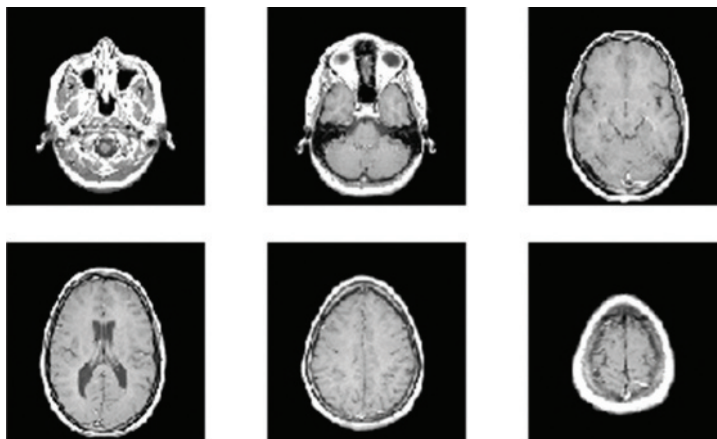


Figure 1.2

Horizontal slices through the brain, based on the sample data file included with MATLAB.



for medical imaging. A limited **MRI** data set that has already been converted to a format compatible with MATLAB ships with the standard MATLAB program. This data set allows you to try out some of the imaging functions available both with the standard MATLAB installation and with the expanded imaging toolbox, if you have it installed on your computer. Figure 1.2 shows six images of horizontal slices through the brain based on the MRI data set.

The same data set can be used to construct a three-dimensional image, such as either of those shown in Figure 1.3. Detailed instructions on how to create these images are included in the MATLAB tutorial, accessed from the help button on the MATLAB toolbar.

1.3.3 Fluid Dynamics

Calculations describing fluid velocities (speeds and directions) are important in a number of different fields. Aerospace engineers in particular are interested in the behavior of gases, both outside an aircraft or space vehicle and inside the combustion chambers. Visualizing the three-dimensional behavior of fluids is tricky, but MATLAB offers a number of tools that make it easier. Figure 1.4 represents the flow-field calculation results for a **thrust-vector control** device as a quiver plot. Thrust-vector

Figure 1.3

Three-dimensional visualization of MRI data, based on the sample data set included with MATLAB.

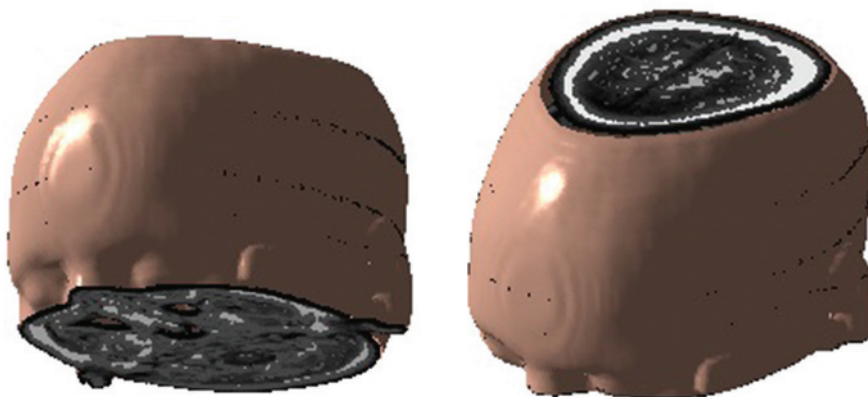
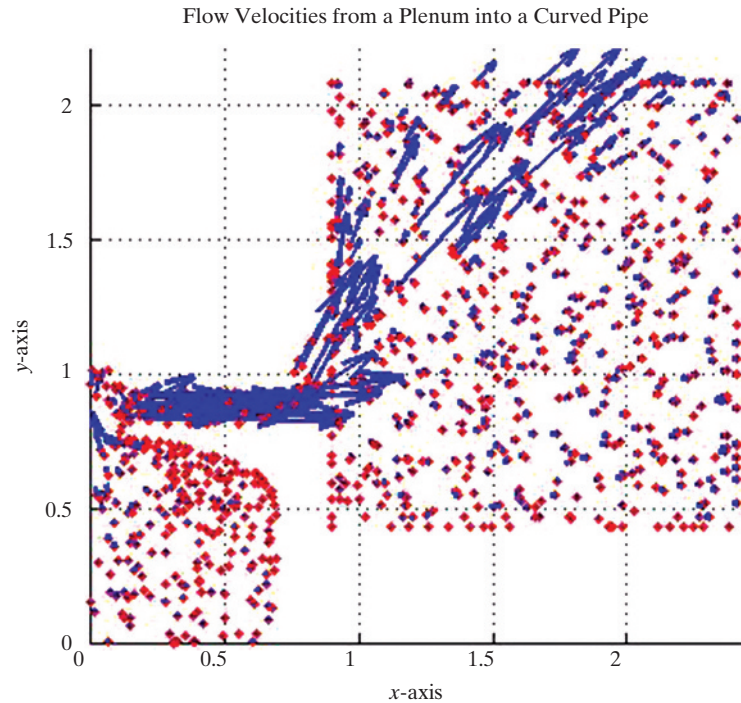


Figure 1.4

Quiver plot of gas behavior in a thrust-vector control device.



control is the process of changing the direction in which a nozzle points (and hence the direction a rocket travels) by pushing on an actuator (a piston-cylinder device). The model in the figure represents a high-pressure reservoir of gas (a plenum) that eventually feeds into the piston and thus controls the length of the actuator.

1.4 PROBLEM SOLVING IN ENGINEERING AND SCIENCE

A consistent approach to solving technical problems is important throughout engineering, science, and computer programming disciplines. The approach we outline here is useful in courses as diverse as chemistry, physics, thermodynamics, and engineering design. It also applies to the social sciences, such as economics and sociology. Different authors may formulate their problem-solving schemes differently, but they all have the same basic format:

KEY IDEA

Always use a systematic problem-solving strategy.

- *State the problem.*
 - Drawing a picture is often helpful in this step.
 - If you do not have a clear understanding of the problem, you are not likely to be able to solve it.
- *Describe the input values (knowns) and the required outputs (unknowns).*
 - Be careful to include units as you describe the input and output values. Sloppy handling of units often leads to wrong answers.
 - Identify constants you may need in the calculation, such as the ideal gas constant and the acceleration due to gravity.
 - If appropriate, label a sketch with the values you have identified, or group them into a table.

- *Develop an **algorithm*** to solve the problem. In computer applications, this can often be accomplished with a *hand example*. You'll need to:
 - Identify any equations relating the knowns and unknowns.
 - Work through a simplified version of the problem by hand or with a calculator.
- *Solve the problem.* In this book, this step involves creating a *MATLAB solution*.
- *Test the solution.*
 - Do your results make sense physically?
 - Do they match your sample calculations?
 - Is your answer really what was asked for?
 - Graphs are often useful ways to check your calculations for reasonableness.

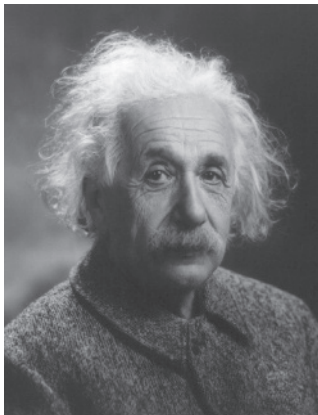
If you consistently use a structured problem-solving approach, such as the one just outlined, you'll find that "story" problems become much easier to solve. Example 1.1 illustrates this problem-solving strategy.

EXAMPLE 1.1

THE CONVERSION OF MATTER TO ENERGY

Figure 1.5

Albert Einstein.
(GL Archive/Alamy Stock
Photo)



Albert Einstein (Figure 1.5) is arguably the most famous physicist of the 20th century. He was born in Germany in 1879 and attended school in both Germany and Switzerland. While working as a patent clerk in Bern, he developed his famous theory of relativity. Perhaps the best-known physics equation today is his

$$E = mc^2.$$

This astonishingly simple equation links the previously separate worlds of matter and energy, and can be used to find the amount of energy released as matter is changed in form in both natural and human-made nuclear reactions.

The sun radiates 385×10^{24} J/s of energy, all of which is generated by nuclear reactions converting matter to energy. Use MATLAB and Einstein's equation to determine how much matter must be converted to energy to produce this much radiation in one day.

1. State the problem
Find the amount of matter necessary to produce the amount of energy radiated by the sun every day.

2. Describe the input and output

Input

Energy: $E = 385 \times 10^{24}$ J/s, which must be converted into the total energy radiated during one day

Speed of light: $c = 3.0 \times 10^8$ m/s

Output

Mass m in kg

3. Develop a hand example
The energy radiated in one day is

$$385 \times 10^{24} \text{ J/s} \times 3600 \text{ s/h} \times 24 \text{ h/day} \times 1 \text{ day} = 3.33 \times 10^{31} \text{ J.}$$

The equation $E = mc^2$ must be solved for m and the values for E and c substituted. Thus,

$$\begin{aligned} m &= \frac{E}{c^2} \\ m &= \frac{3.33 \times 10^{31} \text{ J}}{(3.0 \times 10^8 \text{ m/s})^2} \\ &= 3.7 \times 10^{14} \text{ J}/(\text{m}^2/\text{s}^2). \end{aligned}$$

We can see from the output criteria that we want the mass in kg, so what went wrong? We need to do one more unit conversion:

$$\begin{aligned} 1 \text{ J} &= 1 \text{ kg m}^2/\text{s}^2 \\ &= 3.7 \times 10^{14} \frac{\text{J}}{\text{m}^2/\text{s}^2} \times \frac{\text{kg m}^2/\text{s}^2}{\text{J}} = 3.7 \times 10^{14} \text{ kg} \end{aligned}$$

4. Develop a MATLAB solution

At this point, you have not learned how to create MATLAB code. However, you should be able to see from the following sample code that MATLAB syntax is similar to that used in most algebraic scientific calculators. MATLAB commands are entered at the prompt (`>>`), and the results are reported on the next line. The code is:

```
>> E=385e24    The user types in this information
E =
    3.8500e+026    This is the computer's response
>> E=E*3600*24
E =
    3.3264e+031
>> c=3e8
c =
    300000000
>> m=E/c^2
m =
    3.6960e+014
```

From this point on, we will not show the prompt when describing interactions in the command window.

5. Test the solution

The MATLAB solution matches the hand calculation, but do the numbers make sense? Anything times 10^{14} is a really large number. Consider, however, that the mass of the sun is 2×10^{30} kg. We can calculate how long it would take to consume the mass of the sun completely at a rate of 3.7×10^{14} kg/day. We have

$$\begin{aligned} \text{Time} &= \frac{\text{Mass of the sun}}{\text{Rate of consumption}} \\ \text{Time} &= \frac{2 \times 10^{30} \text{ kg}}{3.7 \times 10^{14} \text{ kg/day}} \times \frac{\text{year}}{365 \text{ days}} = 1.5 \times 10^{13} \text{ years.} \end{aligned}$$

That's 15 trillion years! We don't need to worry about the sun running out of matter to convert to energy in our lifetimes.

KEY TERMS

algorithm	high-level computer	site license
application program	language	surface plasmon
avionics	MRI	thrust vector control
electromagnetic fields	multiparadigm	toolbox
function	operating system	web browser
graphical user interface	script	word-processing program



CHAPTER

2

MATLAB Environment

Objectives

After reading this chapter, you should be able to:

- Start the MATLAB program and solve simple problems in the command window.
- Identify and use the various MATLAB windows.
- Identify and use the basic data types used in MATLAB.
- Define and use simple arrays.
- Name and use variables.
- Use the correct order of operation rules to solve problems using MATLAB.
- Identify the differences between scalar, array, and matrix calculations in MATLAB.
- Express numbers in either floating-point or scientific notation.
- Adjust the format used to display numbers in the command window.
- Save the value of variables used in a MATLAB session.
- Save a series of commands in a MATLAB program, either as a script (M-file) or as a live script (MLX-file).
- Use Section mode.

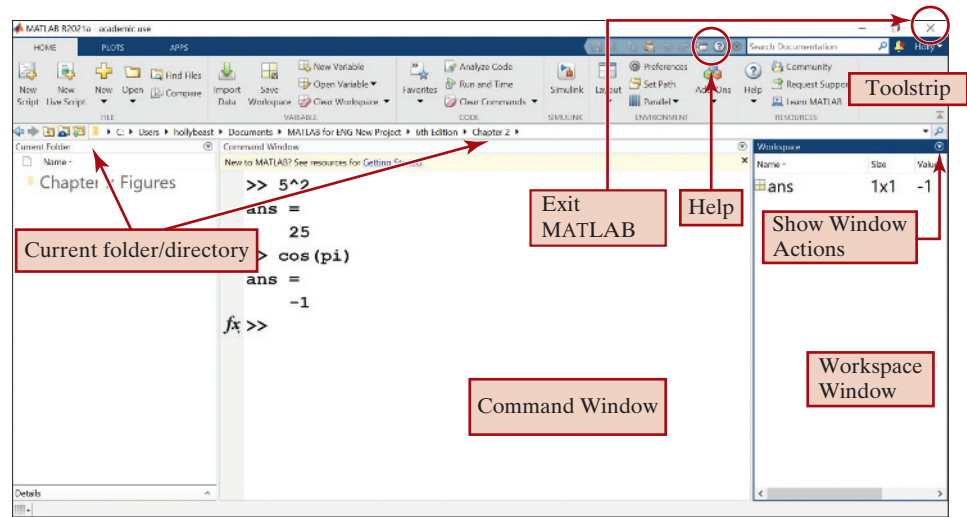
2.1 GETTING STARTED

Using MATLAB for the first time is easy; mastering it can take years. In this chapter, we will introduce you to the MATLAB environment and show you how to perform basic mathematical computations. After reading this chapter, you should be able to start using MATLAB for homework assignments or on the job. Of course, you will be able to do more as you complete the rest of the chapters.

Because the procedure for installing MATLAB depends upon your operating system and your computing environment, we will assume that you have already installed MATLAB on your computer, that you are working in a computing laboratory with MATLAB already installed, or are using MATLAB online. To start MATLAB in either the Windows or Mac environment, double-click on the icon on the desktop, or use

Figure 2.1

MATLAB 2021a opening window. The MATLAB environment consists of a number of windows, three of which open in the default view. Others open as needed during a MATLAB session.



the start menu to find the program. In the UNIX environment, type `Matlab` at the shell **prompt**. No matter how you start it, once MATLAB opens, you should see the MATLAB prompt (`>>` or `EDU>>`), which tells you MATLAB is ready for you to enter a command. When you have finished your MATLAB session, you can exit MATLAB by typing `quit` or `exit` at the MATLAB prompt. MATLAB also uses the standard Windows menu bar, so you can exit the program by selecting the close icon (**x**) at the upper right-hand corner of the screen. The default MATLAB screen, which opens each time you start the program, is shown in Figure 2.1. (There are minor differences in the MATLAB desktop, depending on the release.)

To start using MATLAB, you only need to be concerned with the command window (in the center of the screen). You can perform calculations in the command window in a manner similar to the way you perform calculations on a scientific calculator. Even most of the syntax is the same. For example, to compute the value of 5 squared, type the command

```
5^2
```

The following output will be displayed:

```
ans =
    25
```

Or, to find the value of $\cos(\pi)$, type

```
cos(pi)
```

which results in the output

```
ans =
    -1
```

KEY IDEA

MATLAB uses the standard algebraic rules for order of operation.

MATLAB uses the standard algebraic rules for order of operation, which becomes important when you chain calculations together. These rules will be discussed in Section 2.3.3. Notice that the value of π is built into MATLAB, so you do not have to enter it yourself.

HINT

You may think some of the examples are too simple to type in yourself—that just reading the material is sufficient. However, you will remember the material better if you both read it and type it!

Before going any further, try Practice Exercise 2.1.

PRACTICE EXERCISE 2.1

Type the following expressions into MATLAB at the command prompt, and observe the results. The correct answers can be found on the Pearson website at <https://www.pearsonhighered.com/moore>.

1. $5 + 2$
2. $5 * 2$
3. $5 / 2$
4. $3 + 2 * (4 + 3)$
5. $2.54 * 8 / 2.6$
6. $6.3 - 2.1045$
7. 3.6^2
8. $1 + 2^2$
9. `sqrt(5)`
10. `cos(pi)`

HINT

You may find it frustrating to learn that when you make a mistake, you cannot just overwrite your command after you have executed it. This occurs because the command window is creating a list of all the commands you have entered. You cannot “un-execute” a command, or “un-create” it. What you can do is enter the command correctly, then execute your new version. MATLAB offers several ways to make this easier for you. One way is to use the arrow keys, usually located on the right-hand side of your keyboard. The up arrow, ↑, allows you to move through the list of commands you have executed. Once you find the appropriate command, you can edit it, then execute your new version.

2.2 MATLAB WINDOWS

MATLAB uses several display windows. The three-column view, shown in Figure 2.1, includes in the middle, a large **command window**; located on the right, the **workspace window**, and located on the left the **current folder window**. In addition, the **command history window**, **document windows**, **graphics windows**, and **editing windows** will automatically open when needed. Each will be described in the sections that follow. MATLAB also includes a built-in help tutorial that can be accessed from the **toolbar** by selecting the question mark icon, as shown in Figure 2.1. To personalize your desktop, you can resize any of these windows, stack them on top of each other, close the ones you are not using, or “undock” them from the desktop by

using the “Show Workspace Actions” menu located in the upper right-hand corner of each window. You can restore the default configuration by selecting Layout on the toolstrip, then selecting Default, or you can add additional windows by selecting Layout and choosing the windows you would like to see.

2.2.1 Command Window

The command window is located in the center pane of the three-column view of the MATLAB screen, as shown in Figure 2.1. The command window offers an environment similar to a scratch pad. Using it allows you to save the values you calculate, but not the commands used to generate those values. If you want to save the command sequence, you will need to use the editing window to create a MATLAB program. MATLAB programs can be stored as either **scripts** (in **M-files**) or **live scripts** (in **MLX-files**). Both approaches are valuable. Before we introduce MATLAB programs, we will concentrate on using the command window.

KEY IDEA

The command window is similar to a scratch pad.

2.2.2 Command History

The command history window records the commands issued in the command window. It does not open in the default view but you can add it to your desktop by selecting Layout → Command History and checking “docked.” The examples in this book will show the Command History in a docked configuration. When you exit MATLAB, or when you issue the `clc` command, the command window is cleared, but the command history retains a list of all your commands. You may clear the command history from the “Show Command History Actions” drop-down menu, located in the upper right-hand corner of the window. If you work on a public computer, as a security precaution, MATLAB’s defaults may be set to clear the history when you exit MATLAB. If you entered the earlier sample commands listed in this book, notice that they are repeated in the command history window. This window is valuable for a number of reasons, including allowing you to review previous MATLAB sessions. It can also be used to transfer commands to the command window. For example, first clear the contents of the command window by typing

```
clc
```

This action clears the command window but leaves the data in the command history window intact. You can transfer any command from the command history window to the command window by double-clicking (which also executes the command), or by clicking and dragging the line of code into the command window. Try double-clicking

```
cos(pi)
```

in the command history window. The command is copied into the command window and executed. It should return

```
ans =  
-1
```

Now click and drag

```
5^2
```

from the command history window into the command window. The command will not execute until you hit Enter, and then you will get the result:

```
ans =  
25
```

KEY IDEA

The command history records all of the commands issued in the command window.

You will find the command history useful as you perform more complicated calculations in the command window.

KEY IDEA

The workspace window lists information describing all the variables created by the program.

2.2.3 Workspace Window

The workspace window keeps track of the **variables** you have defined as you execute commands in the command window. These variables represent values stored in the computer memory, which are available for you to use. If you have been doing the examples, the workspace window should show just one variable, `ans`, and indicate that it has a value of 25, and is a double **array**:

Name	Size	Class	Value
ans	1 × 1	double	25

(Your view of the workspace window may be slightly different, depending on how your installation of MATLAB is configured.)

Set the workspace window to show more about the displayed variables by right-clicking on the bar with the column labels. Check `bytes`, in addition to `name`, `value`, `class`, and `size`. Your workspace window should now display the following information, although you may need to resize the window to see all the columns:

Name	Size	Class	Bytes	Value
ans	1 × 1	double	8	25

KEY IDEA

The default data type is double-precision floating-point numbers stored in an array.

The yellow grid-like symbol indicates that the variable `ans` is an array. The size, `1 × 1`, tells us that it is a single value (one row by one column) and therefore a **scalar**. The array uses 8 bytes of memory. MATLAB was written in C, and the **class** designation tells us that in the C language, `ans` is a double-precision floating-point array. For our needs, it is enough to know that the variable `ans` can store a floating-point number (a number with a decimal point). MATLAB considers every number you enter to be a floating-point number, whether you insert a decimal point or not.

In addition to information about the size of the arrays and type of data stored in them, you can also choose to display statistical information about the data. Once again, right-click the bar in the workspace window that displays the column headings. Notice that you can select from a number of different statistical measures, such as the max, min, and standard deviation.

You can define additional variables in the command window, and they will be listed in the workspace window. For example, typing

A = 5

returns

A =
5

Notice that the variable `A` has been added to the workspace window, which lists variables in alphabetical order. Variables beginning with uppercase letters are listed first, followed by variables starting with lowercase letters.

Name	Size	Class	Bytes	Value
A	1 × 1	double	8	5
ans	1 × 1	double	8	25

In Section 2.3.2, we will discuss in detail how to enter arrays into MATLAB. For now, you can enter a simple one-dimensional array by typing

```
B = [1, 2, 3, 4]
```

This command returns

```
B =  
    1  2  3  4
```

The commas are optional; you would get the same result with

```
B = [1 2 3 4]  
  
B =  
    1  2  3  4
```

Notice that the variable B has been added to the workspace window, and that it is a 1 × 4 array:

Name	Size	Class	Bytes	Value
A	1 × 1	double	8	5
ans	1 × 1	double	8	25
B	1 × 4	double	32	[1, 2, 3, 4]

You can define two-dimensional arrays in a similar fashion. Semicolons are used to separate rows. For example,

```
C = [1 2 3 4; 10 20 30 40; 5 10 15 20]
```

returns

```
C =  
    1     2     3     4  
   10    20    30    40  
    5    10    15    20
```

Name	Size	Class	Bytes	Value
A	1 × 1	double	8	5
ans	1 × 1	double	8	25
B	1 × 4	double	32	[1, 2, 3, 4]
C	1 × 4	double	96	<3 × 4 double>

Notice that C appears in the workspace window as a 3 × 4 array. To conserve space, the values stored in the array are not listed.

You can recall the values for any variable by typing in the variable name. For example, entering

```
A
```

returns

```
A =  
    5
```

Although the only variables we have introduced are arrays containing numbers, other types of variables are possible.

In describing the command window, we introduced the `clc` command. This command clears the command window, leaving a blank page for you to work on.

However, it does not delete from memory the actual variables you have created. The `clear` command deletes all of the saved variables, which is reflected in the workspace window. Try it out by typing

`clear`

in the command window. The workspace window is now empty:

Name	Value	Size	Bytes	Class
------	-------	------	-------	-------

If you suppress the workspace window (closing it either from the Layout menu on the toolbar, or from the Window Action menu in the upper right-hand corner of the window), you can still find out which variables have been defined by using the `whos` command:

`whos`

If executed before we entered the `clear` command, `whos` would have returned

Name	Size	Bytes	Class
A	1 × 1	8	double
B	1 × 4	32	double
C	1 × 4	96	double
ans	1 × 1	8	double

2.2.4 Current Folder Window

The current folder window lists all the files in the active directory. When MATLAB either accesses files or saves information, it uses the current folder (also called the current directory) unless told differently. The default for the location of the current folder varies with your version of the software and the way it was installed. However, the current folder is listed at the top of the main window. The current folder can be changed by selecting another directory from the drop-down list located next to the directory listing, or by browsing through your computer files. Browsing is performed with the browse button, located next to the drop-down list (see Figure 2.2).

2.2.5 Document Window

Double-clicking on any variable listed in the workspace window automatically launches a document window, containing the variable editor. Values stored in the variable are displayed in a spreadsheet format. You can change values in the variable editor, or you can add new values. For example, if you have not already entered the two-dimensional array `C`, enter the following command in the command window:

`C = [1 2 3 4; 10 20 30 40; 5 10 15 20];`

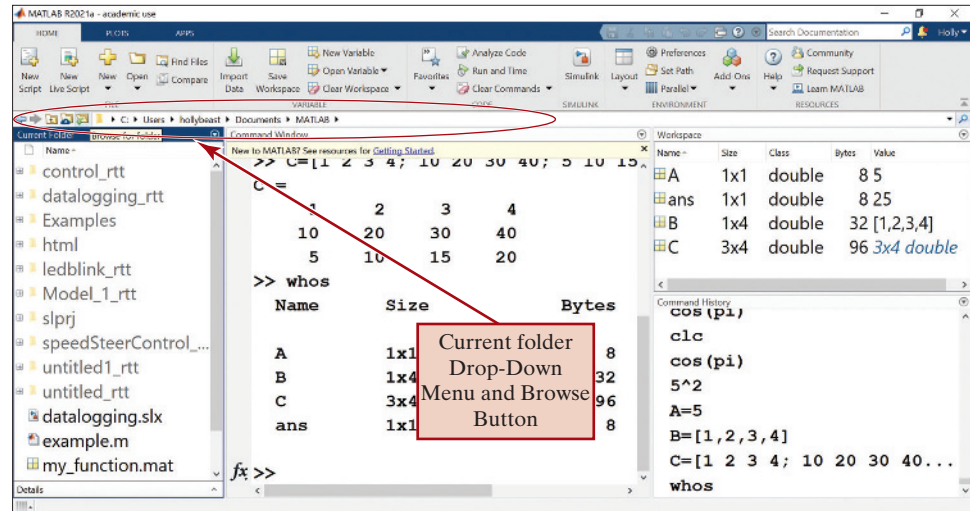
KEY IDEA
A semicolon suppresses the output from commands issued in the command window.

Placing a semicolon at the end of the command suppresses the output so that it is not repeated in the command window. However, `C` should now be listed in the workspace window. If you double-click on it, a document window will open above the command window, as shown in Figure 2.3. You can now add more values to the `C` array or change existing values. Notice that the toolbar automatically opens the variable tab, which includes options that allow you to manipulate the data in the editor.

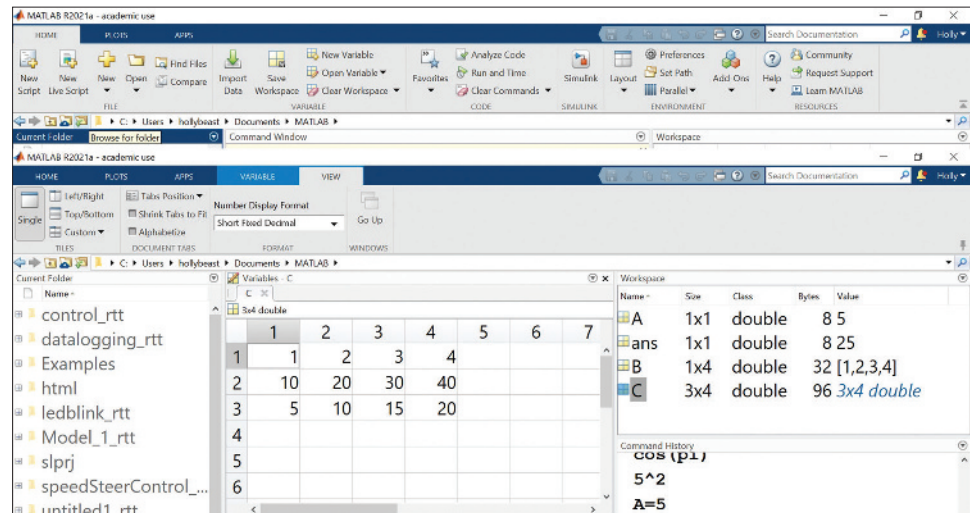
The document window/**array editor** can also be used in conjunction with the workspace window action menu to create entirely new arrays. Right-click on the workspace title bar, select `new` from the menu, and a new variable called `unnamed` should

Figure 2.2

The current folder window lists all the files in the active directory. You can change the current folder by using the drop-down menu or the browse button.

**Figure 2.3**

The document window displays the *variable editor*.



appear on the variable list. You can change its name by right-clicking and selecting *rename* from the pop-up menu. To add values to this new variable, double-click on it and add your data from the variable editor window. When you are finished creating new variables, close the array editor by selecting the close window icon in the upper right-hand corner of the window.

2.2.6 Graphics Window

The graphics window launches automatically when you request a graph. To demonstrate this feature, first create an array of *x* values in the command window:

```
x = [1 2 3 4 5];
```

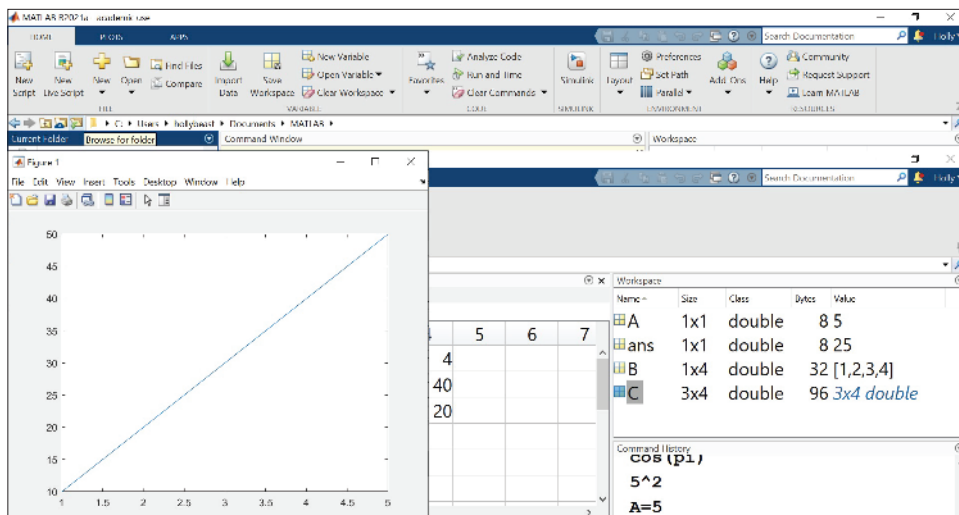
(The semicolon suppresses the output from this command; however, a new variable, *x*, appears in the workspace window.)

Now create a list of *y* values:

```
y = [10 20 30 40 50];
```


Figure 2.4

MATLAB makes it easy to create graphs. The Graphics Window opens on top of the desktop, and is not automatically docked.



To create a graph, use the plot command:

plot(x,y)

The graphics window opens automatically (see Figure 2.4), but is not docked with the main desktop. Notice that a new MATLAB window appears on the task bar at the bottom of the screen. Any additional graphs you create will overwrite Figure 1, unless you specifically command MATLAB to open a new graphics window. You can dock the figure by selecting the arrow in the upper right-hand corner of the window.

KEY IDEA

Always add a title and axis labels to graphs.

MATLAB makes it easy to modify graphs by adding titles, x and y labels, multiple lines, and so on. Annotating graphs will be covered in a separate chapter on plotting. Engineers and scientists **never** present a graph without labels!

2.2.7 Edit Window

The MATLAB editors allow you to type and save a series of commands without executing them. Much of your work in MATLAB will be done in an editing window. There are two types of MATLAB editors: the script editor and the live script editor. To open either editor, choose New Script or New Live Script from the toolstrip. You may also open the edit window by typing `edit` at the command prompt. Notice that the editor tab automatically opens on the toolstrip. Live Scripts are similar to regular scripts, but allow you to include formatted text in your documents. The computer code created in either editor is exactly the same.

2.3 SOLVING PROBLEMS WITH MATLAB

The command window and editing window environments are powerful tools for solving engineering problems. To use them effectively, you will need to understand more about how MATLAB works.

2.3.1 Using Variables

Although you can solve many problems by using MATLAB like a calculator, it is usually more convenient to give names to the values you are using. MATLAB uses the naming conventions that are common to most computer programs:

- All names must start with a letter. The names can be of any length, but only the first 63 characters are used in MATLAB. (Use the `namelengthmax` command

KEY IDEA

Variable names should be short and meaningful.

to confirm this.) Although MATLAB will let you create long variable names, excessive length creates a significant opportunity for error. A common guideline is to use lowercase letters and numbers in variable names, and to use uppercase letters for the names of constants. However, if a constant is traditionally expressed as a lowercase letter, feel free to follow that convention. For example, in physics textbooks, the speed of light is always lowercase *c*. Names should be short enough to remember and should be descriptive.

- The only allowable characters are letters, numbers, and the underscore. You can check to see if a variable name is allowed by using the `isvarname` command. As is standard in computer languages, the number 1 means that something is true, and the number 0 means false. Hence,

```
isvarname time
ans =
    1
```

indicates that `time` is a legitimate variable name, and

```
isvarname cool-beans
ans =
    0
```

tells us that `cool-beans` is not a legitimate variable name. (Recall that the dash is not an allowed character.)

- Names are case sensitive. The variable `x` is different from the variable `X`.
- MATLAB reserves a list of keywords for use by the program, which you cannot assign as variable names. The `iskeyword` command causes MATLAB to list these reserved names:

```
iskeyword
ans =
    20x1 cell array
    {'break'      }
    {'case'       }
    {'catch'      }
    {'classdef'   }
    {'continue'   }
    {'else'       }
    {'elseif'     }
    {'end'        }
    {'for'        }
    {'function'   }
    {'global'     }
    {'if'         }
    {'otherwise'  }
    {'parfor'     }
    {'persistent' }
    {'return'     }
    {'spmd'       }
    {'switch'     }
    {'try'        }
    {'while'      }
```

KEY IDEA

Variable names must start with a letter.

- MATLAB allows you to reassign built-in **function** names as variable names. For example, you could create a new variable called `sin` with the command

```
sin = 4
```

which returns

```
sin =  
4
```

This is clearly a dangerous practice, since the `sin` (i.e., sine) function is no longer available. If you try to use the overwritten function, you'll get an error statement:

```
sin(3)  
Index exceeds the number of array elements (1).
```

and a message with a suggested solution.

`sin` appears to be both a function and a variable. If this is unintentional, use `clear sin` to remove the variable `sin` from the workspace.

You can reset `sin` back to a function by typing

```
clear sin
```

Once you have reset `sin` you can check again to see if it is a function or a variable.

Now when you ask

```
which sin
```

the response is

```
built-in (C:\ProgramFiles\MATLAB\R2011a\toolbox\matlab\elfun\  
@double\sin)  
% double method
```

which tells us the location of the built-in function.

PRACTICE EXERCISE 2.2

Which of the following names are allowed in MATLAB? Make your predictions, then test them with the `isvarname`, `iskeyword`, and `which` commands. The correct answers can be found on the Pearson website at <https://www.pearsonhighered.com/moore>.

1. test
2. Test
3. if
4. my-book
5. my_book
6. Thisisoneverylongnamebutisitstillallowed?
7. 1stgroup
8. group_one
9. zzaAbc
10. z34wAwy?12#
11. sin
12. log

KEY IDEA

All data in MATLAB is stored in arrays.

KEY IDEA

MATLAB arrays can be scalars (a single value), vectors (a row or column), two-dimensional arrays or multi-dimensional arrays.

KEY IDEA

MATLAB arrays can hold numeric as well as other types of information.

2.3.2 Data Types

The basic **data type** used in MATLAB is the array. A single value, called a scalar, is represented as a 1×1 array. A list of values, arranged in either a column or a row, is a one-dimensional array called a **vector**. A table of values is represented as a two-dimensional array. Although we'll limit ourselves to scalars, vectors, and two-dimensional arrays in this chapter, MATLAB can handle higher order arrays. The terms **matrix** and array are used interchangeably by most MATLAB users, even though they are technically different in a mathematical context. An array is really only a matrix if it is used in matrix algebra calculations, which are discussed in a later chapter.

The array data type is further divided into categories, which in MATLAB are confusingly also simply called data types, or classes. Thus far we have used MATLAB with numeric information. Numbers in MATLAB default to the double-precision floating-point representation (`double`), which is only one of the possible numeric data types available. Other less commonly used numeric data types are discussed in later chapters. Another kind of data that is widely used in engineering and scientific applications is textual information. MATLAB offers two data types for storing text: character arrays (`char`) and string arrays (`string`). Logical arrays store information about whether things are true or false and are useful in making decisions in programming situations. There are many other kinds of MATLAB arrays that store just a single type of information, but these four basic types (doubles, character, strings, and logical arrays) are enough for a beginning programmer to get started.

There are also data types that can store multiple data types in the same array. The most useful for beginners is the `table`, especially because it is the default data type when data is imported from files. Tables look a lot like spreadsheets. They are used to both store information and to display results.

The basic rules for manipulating information in arrays are the same, independent of the data type. In this chapter, calculations with numeric arrays will be explored. The next few chapters will introduce textual information, logical information, and tables. Later chapters will explore the wide variety of data types available for you to use, once you have mastered basic programming concepts.

2.3.3 Calculations with Numeric Arrays

In mathematical nomenclature, arrays are represented as rows and columns inside square brackets:

$$A = [5] \quad B = [2 \quad 5] \quad C = \begin{bmatrix} 1 & 2 \\ 5 & 7 \end{bmatrix}$$

In this example, A is a 1×1 array, B is a 1×2 array, and C is a 2×2 array. The advantage in using array representation is that whole groups of information can be represented with a single name. Most people feel more comfortable assigning a name to a single value, so we'll start by explaining how MATLAB handles scalars, then move on to more complicated arrays.

Scalar Operations

MATLAB handles arithmetic operations between two scalars much the same as other computer programs and handheld calculators. The syntax for addition, subtraction, multiplication, division, and exponentiation is shown in Table 2.1. The command

```
a = 1 + 2
```

Table 2.1 Arithmetic Operations Between Two Scalars (Binary Operations)

Operation	Algebraic Syntax	MATLAB Syntax
Addition	$a + b$	a + b
Subtraction	$a - b$	a - b
Multiplication	$a \times b$	a * b
Division	$\frac{a}{b}$ or $a \div b$	a / b
Exponentiation	a^b	a ^ b

should be read as “a is assigned a value of 1 plus 2,” which is the addition of two scalar quantities. Arithmetic operations between two scalar variables use the same syntax. Suppose, for example, that you have defined a in the previous statement and that b has a value of 5:

```
b = 5  
  
Then  
  
x = a + b
```

returns the following result:

```
x =  
  
8
```

A single equals sign (=) is called an **assignment operator** in MATLAB. The assignment operator causes the result of your calculations to be stored in a computer memory location. In the preceding example, x is assigned a value of 8. If you enter the variable name

```
x
```

into MATLAB, you get the following result:

```
x =  
  
8
```

KEY IDEA
The assignment operator is different from an equality.

The assignment operator is significantly different from an equality. Consider the statement

```
x = x + 1
```

This is not a valid algebraic statement, since x is clearly not equal to x + 1. However, when interpreted as an assignment statement, it tells us to replace the current value of x stored in memory with a new value that is equal to the old x plus 1.

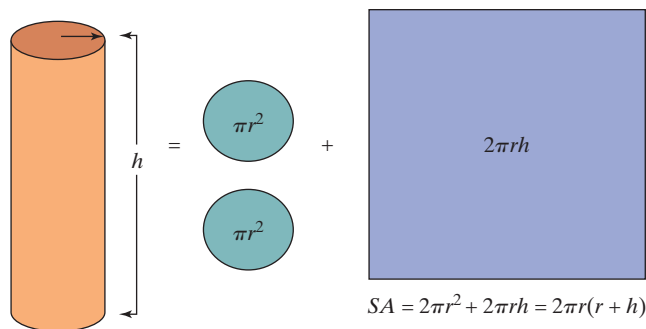
Since the value stored in x was originally 8, the statement returns

```
x =  
  
9
```

indicating that the value stored in the memory location named x has been changed to 9. The assignment statement is similar to the process of saving a file. When you first save a word-processing document, you assign it a name. Subsequently, after you’ve made changes, you resave your file, but still assign it the same name. The first and second versions are not equal: You’ve just assigned a new version of your document to an existing memory location.

Figure 2.5

Finding the surface area of a right circular cylinder involves addition, multiplication, and exponentiation.



Order of Operations

In all mathematical calculations, it is important to understand the order in which operations are performed. MATLAB follows the standard algebraic rules for the order of operation:

- First, perform calculations inside parentheses, working from the innermost set to the outermost.
- Next, perform exponentiation operations.
- Then, perform multiplication and division operations, working from left to right.
- Finally, perform addition and subtraction operations, working from left to right.

To better understand the importance of the order of operations, consider the calculations involved in finding the surface area of a right circular cylinder.

The surface area is the sum of the areas of the two circular bases and the area of the curved surface between them, as shown in Figure 2.5. If we let the height of the cylinder be 10 cm and the radius 5 cm, the following MATLAB code can be used to find the surface area:

```
radius = 5;
height = 10;
surface_area = 2*pi*radius^2 + 2*pi*radius*height
```

The code returns

```
surface_area =
    471.2389
```

In this case, MATLAB first performs the exponentiation, raising the radius to the second power. It then works from left to right, calculating the first product and then the second product. Finally, it adds the two products together. You could instead formulate the expression as

```
surface_area = 2*pi*radius*(radius + height)
```

which also returns

```
surface_area =
    471.2389
```

In this case, MATLAB first finds the sum of the radius and height then performs the multiplications, working from left to right. If you forgot to include the parentheses, you would have

```
surface_area = 2*pi*radius*radius + height
```

in which case the program would have first calculated the product of $2\pi r^2$ and then added height—resulting in the wrong answer.

Note that it was necessary to include the multiplication operator before the parentheses, because MATLAB does not assume any operators and would misinterpret the expression

```
radius(radius + height)
```

as follows. The value of radius plus height is 15 (radius = 10 and height = 5) so MATLAB would have looked for the 15th value in an array called radius. This interpretation would have resulted in the following error statement:

```
??? Index exceeds the number of array elements (1).
```

It is important to be extra careful in converting equations into MATLAB statements. There is no penalty for adding extra parentheses, and they often make the code easier to interpret, both for the programmer and for others who may use the code in the future. Here's another common error that could be avoided by liberally using parentheses. Consider the following mathematical expression:

$$\frac{-Q}{e^{RT}}$$

In MATLAB, the mathematical constant e is evaluated using the function, `exp`, so the appropriate syntax is

```
exp(-Q/(R*T))
```

Unfortunately, leaving out the parentheses as in

```
exp(-Q/R*T)
```

gives a very different result. Since the expression is evaluated from left to right, first Q is divided by R , then the result is multiplied by T —not at all what was intended.

Another way to make computer code more readable is to break long expressions into multiple statements. For example, consider the equation

$$f = \frac{\log(ax^2 + bx + c) - \sin(ax^2 + bx + c)}{4\pi x^2 + \cos(x - 2)(ax^2 + bx + c)}$$

It would be very easy to make an error keying in this equation. To minimize the chance of that happening, break the equation into several pieces. For example, first assign values for x , a , b , and c (you can do it all on the same line):

```
x = 9; a = 1; b = 3; c = 5;
```

Then define a polynomial and the denominator:

```
poly = a*x^2 + b*x + c;
denom = 4*pi*x^2 + cos(x - 2)*poly;
```

Combine these components into a final equation:

```
f = (log(poly) - sin(poly))/denom
```

The result is

```
f =
    0.0044
```

This approach minimizes your opportunity for error. Instead of keying in the polynomial three times (and risking an error each time), you need to key it in only once. Your MATLAB code is more likely to be accurate, and it's easier for others to understand.

KEY IDEA

Try to minimize opportunity for error.

HINT

MATLAB does not read “white space,” so you may add spaces to your commands without changing their meaning. A long expression is easier to read if you add a space before and after plus (+) signs and minus (−) signs but not before and after multiplication (*) and division (/) signs.

PRACTICE EXERCISE 2.3

Predict the results of the following MATLAB expressions, then check your predictions by keying the expressions into the command window. The correct answers can be found on the Pearson website at <https://www.pearson-highered.com/moore>.

1. $6/6 + 5$
2. $2*6^2$
3. $(3 + 5)*2$
4. $3 + 5*2$
5. $4 * 3/2 * 8$
6. $3 - 2/4 + 6^2$
7. 2^3*4
8. $2^(3^4)$
9. 3^5+4
10. $3^(5+4)$

Create and test MATLAB syntax to evaluate the following expressions, and then check your answers with a handheld calculator.

11. $\frac{5 + 3}{9 - 1}$
12. $2^3 - \frac{4}{5 + 3}$
13. $\frac{5^{2+1}}{4 - 1}$
14. $4\frac{1}{2} * 5\frac{2}{3}$
15. $\frac{5 + 6 * \frac{7}{3} - 2^2}{\frac{2}{3} * \frac{3}{3 * 6}}$

EXAMPLE 2.1**SCALAR OPERATIONS**

Wind tunnels (see Figure 2.6) play an important role in our study of the behavior of high-performance aircraft. In order to interpret wind tunnel data, engineers need to understand how gases behave. The basic equation describing the properties of gases is the ideal gas law, a relationship studied in detail in freshman chemistry classes. The law states that

Figure 2.6

Wind tunnels are used to test aircraft designs. Kitty O'Brien Joyner was an American Electrical Engineer with the National Advisory Committee for Aeronautics (NACA) and then with NASA. She was the first woman engineer at NASA and managed several of its wind tunnels. She is shown analyzing the operation of a wind tunnel turbine at NACA Langley in 1952. (Courtesy of NASA)



$$PV = nRT$$

where P = pressure in kPa,

V = volume in m^3 ,

n = number of kmoles of gas in the sample,

R = ideal gas constant, $8.314 \text{ kPa m}^3/\text{kmol K}$, and

T = temperature, expressed in kelvins (K).

In addition, we know that the number of kmoles of gas is equal to the mass of the gas divided by the molar mass (also known as the molecular weight) or

$$n = m/MW$$

where

m = mass in kg and

MW = molar mass in kg/kmol .

Different units can be used in the equations if the value of R is changed accordingly.

Now, suppose you know that the volume of air in the wind tunnel is 1000 m^3 . Before the wind tunnel is turned on, the temperature of the air is 300 K , and the pressure is 100 kPa . The average molar mass (molecular weight) of air is approximately $29 \text{ kg}/\text{kmol}$. Find the mass of the air in the wind tunnel.

To solve this problem, use the following problem-solving methodology:

1. State the problem

When you solve a problem, it is a good idea to restate it in your own words: Find the mass of air in a wind tunnel.

2. Describe the input and output

Input

Volume	$V = 1000 \text{ m}^3$
Temperature	$T = 300 \text{ K}$
Pressure	$P = 100 \text{ kPa}$
Molecular weight	$MW = 29 \text{ kg}/\text{kmol}$
Gas constant	$R = 8.314 \text{ kPa m}^3/\text{kmol K}$

Output

Mass $m = ? \text{ kg}$

(continued)

3. Develop a hand example

Working the problem by hand (or with a calculator) allows you to outline an algorithm, which you can translate to MATLAB code later. You should choose simple data that make it easy to check your work. In this problem, we know two equations relating the data:

$$PV = nRT \quad \text{ideal gas law}$$

$$n = m/MW \quad \text{relationship between mass and moles}$$

Solve the ideal gas law for n , and plug in the given values:

$$\begin{aligned} n &= PV/RT \\ &= \frac{100 \text{ kPa} \times 1000 \text{ m}^3}{8.314 \text{ kPa m}^3/\text{kmol K} \times 300\text{K}} \\ &= 40.0930 \text{ kmol} \end{aligned}$$

Convert moles to mass by solving the conversion equation for the mass m and plugging in the values:

$$\begin{aligned} m &= n \times MW = 40.0930 \text{ kmol} \times 29 \text{ kg/mol} \\ m &= 1162.70 \text{ kg} \end{aligned}$$

4. Develop a MATLAB solution

First, clear the screen and memory:

```
clear, clc
```

Now perform the following calculations in the command window:

```
P = 100
P =
    100
T = 300
T =
    300
V = 1000
V =
    1000
MW = 29
MW =
    29
R = 8.314
R =
    8.3140
n = (P*V) / (R*T)
n =
    40.0930
m = n*MW
m =
    1.1627e+003
```

There are several things you should notice about this MATLAB solution. First, because no semicolons were used to suppress the output, the values of the variables are repeated after each assignment statement. Notice also the use of parentheses in the calculation of n . They are necessary in the denominator, but not in the numerator. However, using parentheses in both makes the code easier to read.

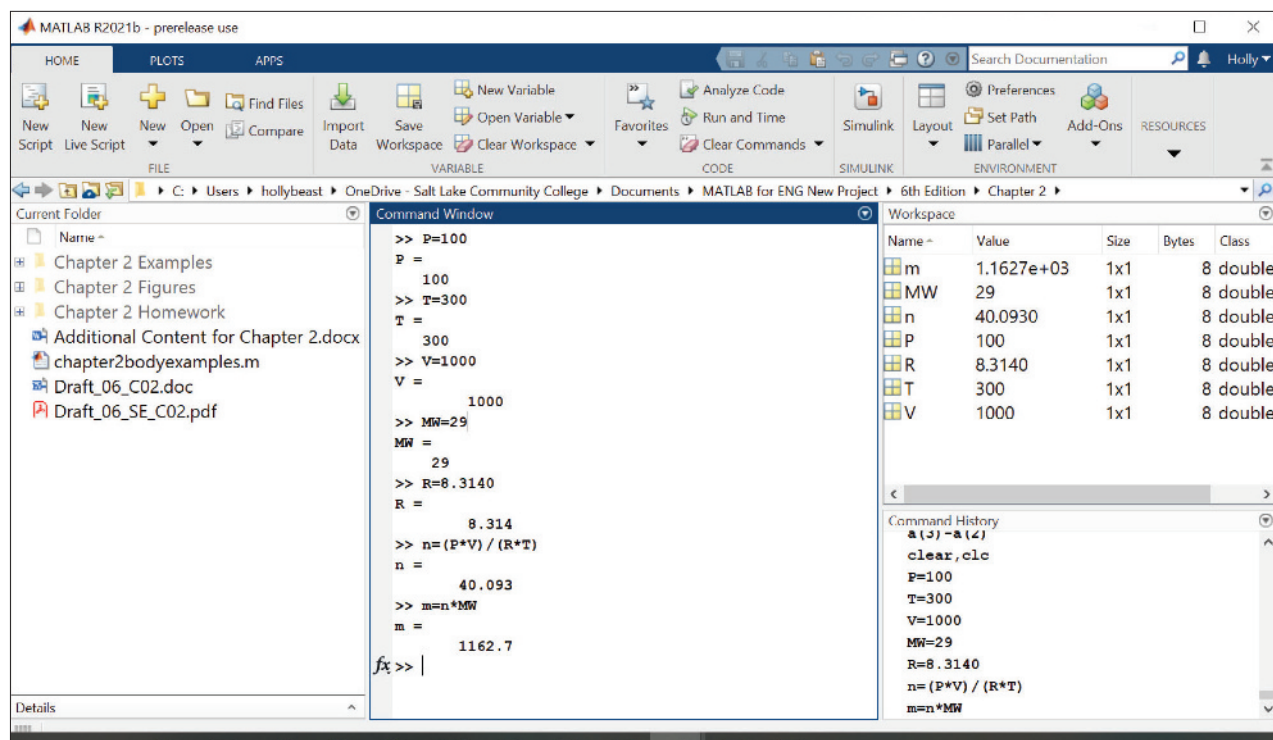


Figure 2.7

MATLAB screen used to solve the ideal gas problem.

5. Test the solution

In this case, comparing the result with that obtained by hand is sufficient. More complicated problems solved in MATLAB should use a variety of input data, to confirm that your solution works in a variety of cases. The MATLAB screen used to solve this problem is shown in Figure 2.7.

You may wonder why you should bother to do a hand calculation if you have MATLAB to do the calculation for you. Once you are proficient with the program, a quick approximation is all that is really necessary. However, new users often make simple mistakes like forgetting to use parentheses in the denominator, such as:

$$n = (P*V)/R*T$$

which results in an incorrect answer. While you are learning, you should check your work with a tool with which you are comfortable, such as your handheld calculator.

Notice that the variables defined in the command window are listed in the workspace window. Notice also that the command history lists the commands executed in the command window. If you were to scroll up in the command history window, you would see commands from previous MATLAB sessions. All of these commands are available for you to move to the command window.

KEY IDEA

MATLAB's strength is in array calculations.

Array Operations

Using MATLAB as a glorified calculator is fine, but its real strength is in array manipulations. As described previously, the simplest way to define an array is to use a list of numbers, called an **explicit list**. The command

```
x = [1 2 3 4]
```

returns the row vector

```
x =  
1 2 3 4
```

Recall that, in defining this vector, you may list the values either with or without commas. A new row is indicated by a semicolon, so a column vector is specified as

```
y = [1; 2; 3; 4]
```

and an array that contains both rows and columns is created with the statement

```
a = [1 2 3 4; 2 3 4 5; 3 4 5 6]
```

and will return

```
a =  
1 2 3 4  
2 3 4 5  
3 4 5 6
```

HINT

It's easier to keep track of how many values you've entered into an array if you enter each row on a separate line—the semicolon is optional.

```
a = [1 2 3 4;  
2 3 4 5;  
3 4 5 6]
```

While a complicated array might have to be entered by hand, evenly spaced arrays can be entered much more readily. The command

```
b = 1:5
```

and the command

```
b = [1:5]
```

are equivalent statements. Both return a row array

```
b =  
1 2 3 4 5
```

(The square brackets are optional.) The default increment is 1, but if you want to use a different increment, put it between the first and final values on the right side of the command. For example,

```
c = 1:2:5
```

indicates that the increment between values will be 2 and returns

```
c =
    1    3    5
```

If you want MATLAB to calculate the spacing between elements, you may use the `linspace` command. Specify the initial value, the final value, and how many total values you want. For example,

```
d = linspace(1, 10, 3)
```

returns a vector with three values, evenly spaced between 1 and 10:

```
d =
    1    5.5   10
```

If you don't specify the number of values in the third field, the function defaults to 100 values, so that

```
d = linspace(1, 10);
```

creates an array with 100 values. In this example a semicolon was used to suppress the output, but by referring to the workspace window you can see that `d` is a 1×100 vector.

You can create logarithmically spaced vectors with the `logspace` command, which also requires three inputs. The first two values are powers of 10 representing the initial and final values in the array. The final value is the number of elements in the array. Thus,

```
e = logspace(1, 3, 3)
```

returns three values:

```
e =
    10   100  1000
```

Notice that the first element in the vector is 10^1 , and the last element in the array is 10^3 .

HINT

New MATLAB users often err when using the `logspace` command by entering the actual first and last values requested, instead of the corresponding power of 10. For example,

```
logspace (10,100,3)
```

is interpreted by MATLAB as: Create a vector from 10^{10} to 10^{100} with three values. The result is

```
ans =
    1.0e+100 *
    0.0000  0.0000  1.0000
```

A common multiplier (1×10^{100}) is specified for each result, but the first two values are so small in comparison to the third, that they are effectively 0.

HINT

You can include mathematical operations inside an array definition statement. For example, you might have $a = [0:\pi/10:\pi]$

Arrays can be used in many calculations with scalars. If

```
a = [ 1 2 3 ]
```

we can add 5 to each value in the array with the syntax

```
b = a + 5
```

which returns

```
b =
    6    7    8
```

KEY IDEA

Matrix multiplication is different from element-by-element (array) multiplication.

This approach works well for addition and subtraction; however, multiplication and division are different. In matrix mathematics, the multiplication operator (*) indicates matrix multiplication. Because all MATLAB operations can involve arrays, we need a different operator to indicate element-by-element multiplication. That operator is .* (called *dot multiplication* or *array multiplication*). For example,

```
a .* b
```

results in element 1 of array a being multiplied by element 1 of array b, element 2 of array a being multiplied by element 2 of array b, element n of array a being multiplied by element n of array b.

For the particular case of our a (which is [1 2 3]) and our b (which is [6 7 8]),

```
a .* b
```

returns

```
ans =
    6   14   24
```

(Do the math to convince yourself that these are the correct answers.)

When you multiply a scalar times an array, you may use either operator (* or .*), but when you multiply two arrays together, they mean something quite different. Just using * implies a matrix multiplication, which in this case would return an error message, because a and b here do not meet the rules for multiplication in matrix algebra. The moral is, be careful to use the correct operator when you mean element-by-element multiplication.

Similar syntax holds for exponentiation (.^) and element-by-element division (./) of individual elements:

```
a.^2
a./b
```

Unfortunately, when you divide a scalar by an array you still need to use the ./ syntax, because the / operator means taking the matrix inverse to MATLAB. As a rule, unless you specifically are doing problems involving linear algebra (matrix mathematics), you should use the **dot operators**.

As an exercise, predict the values resulting from the preceding two expressions, then test your predictions by executing the commands in MATLAB.

PRACTICE EXERCISE 2.4

As you perform the following calculations, recall the difference between the `*` and `.*` operators, as well as the `/` and `./` and the `^` and `.^` operators. The correct answers can be found on the Pearson website at <https://www.pearsonhighered.com/moore>.

1. Define the array $a = [2.3 \ 5.8 \ 9]$ as a MATLAB variable.
2. Find the sine of a .
3. Add 3 to every element in a .
4. Define the array $b = [5.2 \ 3.14 \ 2]$ as a MATLAB variable.
5. Add together each element in array a and in array b .
6. Multiply each element in a by the corresponding element in b .
7. Square each element in array a .
8. Create an array named c of evenly spaced values from 0 to 10, with an increment of 1.
9. Create an array named d of evenly spaced values from 0 to 10, with an increment of 2.
10. Use the `linspace` function to create an array of six evenly spaced values from 10 to 20.
11. Use the `logspace` function to create an array of five logarithmically spaced values between 10 and 100.

KEY IDEA

The array capability of MATLAB makes it easy to do repetitive calculations.

The array capability of MATLAB makes it easy to do repetitive calculations. For example, suppose you have a list of angles in degrees that you would like to convert to radians. First put the values into an array. For angles of 10, 15, 70, and 90, enter

```
degrees = [10 15 70 90];
```

To change the values to radians, you must multiply by $\pi/180$:

```
radians = degrees*pi/180
```

This command returns an array called `radians`, with the values in radians. (Try it!) In this case, you could use either the `*` or the `.*` operator, because the multiplication involves a single array (`degrees`) and two scalars (`pi` and 180). Thus, you could have written

```
radians = degrees.*pi/180
```

HINT

The value of π is built into MATLAB as a floating-point number called `pi`.

Because π is an irrational number, it cannot be expressed *exactly* with a floating-point representation, so the MATLAB constant `pi` is really an approximation. You can see this when you find `sin(pi)`. From trigonometry, the answer should be 0. However, MATLAB returns a very small number, 1.2246e-016. In most calculations, this won't make a difference in the final result.