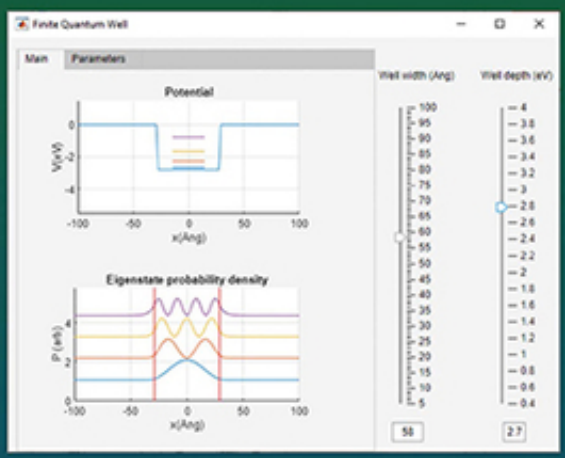
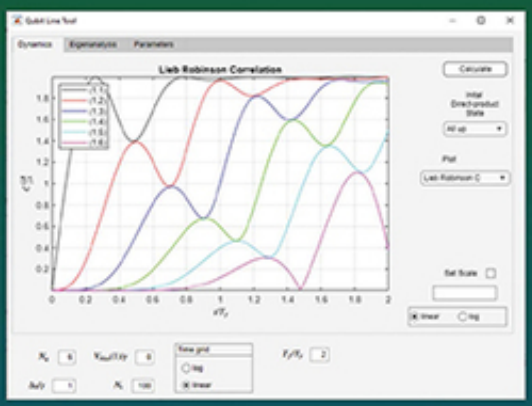
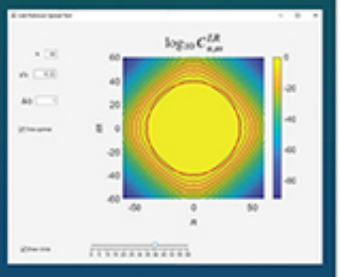
Second Edition

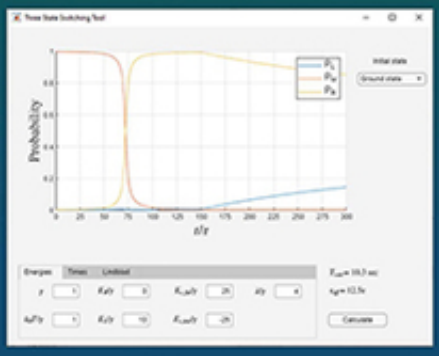
Learning to Program with MATLAB

Building GUI Tools









with website

Craig S. Lent

WILEY

LEARNING TO PROGRAM WITH MATLAB

LEARNING TO PROGRAM WITH MATLAB

Building GUI Tools

Second Edition

Craig S. Lent
University of Notre Dame
Notre Dame, USA

WILEY

This edition first published 2022

Edition History
John Wiley & Sons (1e, 2013)

Registered Office
John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, USA

Editorial Office
111 River Street, Hoboken, NJ 07030, USA

For details of our global editorial offices, customer services, and more information about Wiley products visit us at www.wiley.com.

Wiley also publishes its books in a variety of electronic formats and by print-on-demand. Some content that appears in standard print versions of this book may not be available in other formats.

Library of Congress Cataloging-in-Publication Data

Names: Lent, Craig S., 1956- author.
Title: Learning to program with MATLAB : building GUI tools / Craig S. Lent, University of Notre Dame, Notre Dame, USA.
Description: Second edition. | Hoboken, NJ : Wiley, 2022. | Includes index.
Identifiers: LCCN 2022017476 (print) | LCCN 2022017477 (ebook) | ISBN 9781119900474 (paperback) | ISBN 9781119900481 (adobe pdf) | ISBN 9781119900498 (epub)
Subjects: LCSH: Computer programming. | Visual programming (Computer science) | MATLAB. | Graphical user interfaces (Computer systems) | BISAC: COMPUTERS / General.
Classification: LCC QA76.6 .L45 2022 (print) | LCC QA76.6 (ebook) | DDC 005.13–dc23/eng/20220527
LC record available at https://lccn.loc.gov/2022017476
LC ebook record available at https://lccn.loc.gov/2022017477

Cover design: Wiley
Cover images: Images courtesy of the author

Set in 10/12pt TimesLTStd by Straive, Chennai, India

To Tom Finke, the late Pat Malone, and Katy McShane, and
all the amazing teachers at the Trinity School campuses in
South Bend, IN, Eagan MN, and Falls Church, VA.

Contents

Preface to the Second Edition

To learn how to program a computer, in a modern language with serious graphical capabilities, is to take hold of a tool of remarkable flexibility that has the power to provide profound insight. This text is primarily aimed at being a first course in programming, and is oriented toward integration with science, mathematics, and engineering. It is also useful for more advanced students and researchers who want to rapidly acquire the ability to easily build useful graphical tools for exploring computational models. The MATLAB programming language provides an excellent introductory language, with built-in graphical, mathematical, and user-interface capabilities. The goal is that the student learns to build computational models with graphical user interfaces (GUIs) that enable exploration of model behavior. This GUI tool-building approach has been used at multiple educational levels: graduate courses, intermediate undergraduate courses, an introductory engineering course for first-year college students, and high school junior and senior-level courses.

The MATLAB programming language, descended from FORTRAN, has evolved to include many powerful and convenient graphical and analysis tools. It has become an important platform for engineering and science education, as well as research. MATLAB is a very valuable first programming language, and for many, will be the preferred language for most, if not all, of the computational work they do. Of course, C++, Java, Python, and many other languages play crucial roles in other domains. Several language features make the MATLAB language easier for beginners than many alternatives: it is interpreted rather than compiled; variable types and array sizes need not be declared in advance; it is not strongly typed; vector, matrix, multidimensional array, and complex numbers are basic data types; there is a sophisticated integrated development and debugging environment; a rich set of mathematical and graphics functions are provided.

While computer programs can be used in many ways, the emphasis here is on building computational models, primarily of physical phenomena (though the techniques can be easily extended to other systems). A physical system is modeled first conceptually, using ideas such as momentum, force, energy, reactions, fields. These concepts are expressed mathematically and applied to a particular class of problem. Such a class might be, for example, projectile motion, fluid flow, quantum evolution, electromagnetic fields, circuit equations, or Newton's laws. Typically, the model involves a set of parameters which describe the physical system and a set of mathematical relations (systems of equations, integrals, differential equations, eigensystems, etc.). The mathematical solution process must be realized through a computational algorithm—a step-by-step procedure for calculating the desired quantities from the input parameters. The behavior of the model is then usually visualized graphically, e.g., one or more plots, bar graphs, or animations.

A GUI tool consists of a computational model and a graphical user interface that lets the user easily and naturally adjust the parameters of the model, rerun the computation, and see the new results.

The experience which led to this text was the observation that student learning is enhanced if the students themselves build the GUI tool: construct the computational model, implement the visualization of results, and design the GUI.

The GUI is valuable for several reasons. The most important is that exploring model behavior, by manipulating sliders, buttons, checkboxes, and the like, encourages a focus on developing an intuitive insight into the model behavior. Insight is the primary goal. Running the model many times with differing inputs, the user can start to see the characteristic behavior of physical system represented by the model. Additionally, it must be recognized that graphically driven tools are what students are accustomed to when dealing with computers. A command line interface seems crude and retrograde. Moreover, particularly for engineering students, the discipline of wrapping the model in a form that someone *else* could use, encourages a design-oriented mentality. Finally, building and delivering a sophisticated mathematical model that is operated through a GUI interface is simply more rewarding and fun.

The GUI tool orientation guides the structure of the text. Part I (Chapters 1–8) covers the fundamentals of MATLAB programming and basic graphics. It is designed to be what one needs to know prior to actual GUI building. The goal is to get the student ready for GUI building as quickly as possible (but not quicker).

In this context, Chapters 4 (matrices) and 6 (animation) warrant comment. Because arrays are a basic MATLAB data class and solving linear systems a frequent application, this material is included in Part I. An instructor could choose to cover it later without disrupting the flow of the course. Similarly, the animation techniques covered in Chapter 6 could be deferred. The animation process does, however, provide very helpful and enjoyable practice at programming `for` loops. Many GUI tools are enhanced by having an animation component; among other advantages, animation provides a first check of model behavior against experience. The end of Chapter 6 also includes a detailed discussion of the velocity Verlet algorithm as an improvement on the Euler method for solving systems governed by Newton's second law. While this could be considered a more advanced topic, without it, models as simple as harmonic motion or bouncing balls fail badly because of nonconservation of energy.

Part II covers GUI tool creation with the App Designer program, which is part of MATLAB. Chapters 9 and 10 are the heart of the text and take a very tutorial approach to GUI building. Chapter 9 details a simple, but widely useful, technique for transforming a functioning MATLAB program into a GUI tool. Readers already familiar with MATLAB, but unfamiliar with using App Designer, can likely work through this chapter in a couple of hours and be in short order making GUI tools.

Part III, Chapters 11–13, covers more advanced techniques for graphics, mathematics, and using files. It is not meant to be comprehensive; the online MATLAB help documentation is excellent and will be the main source for a lot of details. The text covers what, in many cases, is the simplest way to invoke a particular function; more complicated uses being left for the student to explore using the documentation.

This approach—having students write GUI tools for specific problem domains—grew out of the author's experience teaching undergraduate electromagnetics courses and graduate quantum mechanics courses in electrical engineering at the University of Notre Dame. These areas are characterized by a high level of mathematical abstraction, so having students transform the esoteric mathematics first into code, and then into visualizable answers, proved invaluable.

The text began as a set of lecture notes for high school students at Trinity School at Greenlawn, in South Bend, Indiana. Since 2005, all Trinity juniors have learned MATLAB using this approach and have used it extensively in the physics and calculus courses that span the junior and senior year. The two other Trinity School campuses, one in Falls Church, Virginia, and the other in Eagan, Minnesota, adopted the curriculum soon after the Greenlawn campus. The chapter on mathematics is largely shaped by the material covered in the Trinity senior year. The author is profoundly grateful to the faculty and students of Trinity Schools, for their feedback, love of learning, and courage. Special thanks to Tom Finke, the remarkable head of Math and Science for Trinity Schools, and to Dr. John Vogel of Trinity School at River Ridge for very helpful reviews of the manuscript.

A note on formatting: numerous examples, programs, and code fragments are included in highlighted text. When the example is meant to illustrate the behavior of MATLAB commands typed in the Command window, the MATLAB command prompt "≫" is included, as in:

```
>> disp("Hello, world!")
Hello, world!
```

Program listings, by contrast, contain the code as it would be seen in the Editor window.

```
%% greetings.m
%  Greet user in cheery way
%     Author: Calvin Manthorn
greeting="Hello, world!";
disp(greeting);
```

After many decades of nearly daily use, the author still finds a durable and surprising joy in writing MATLAB programs for research, teaching, and recreation. It is hoped that, through all the details of the text, this comes through. May you, too, enjoy.

Craig S. Lent, University of Notre Dame, Notre Dame, IN, June 2022

About the Companion Website

This book is accompanied by a companion website:

www.wiley.com/go/learningtoprogramwithmatlab2e

The website includes MATLAB codes from the text by chapter.

MATLAB Programming

PART
I

Getting Started

CHAPTER 1

This chapter will introduce the basics of using MATLAB, first as a powerful calculator, and then as a platform for writing simple programs that automate what a calculator would do in many steps. The emphasis here will be on performing basic mathematical operations on numbers.

The MATLAB integrated development environment (IDE) is the program that runs when you launch MATLAB. You will use it to operate MATLAB interactively and to develop and run MATLAB programs.

The concept of a MATLAB variable is important to grasp. It is not identical with the familiar mathematical notion of a variable, although the two are related. MATLAB variables should be thought of as labeled boxes that hold a number or other types of information.

MATLAB has many built-in functions for evaluating common mathematical functions. More complicated MATLAB functions, including those of your own making, will be explored further in Chapter 7.

After completing this chapter you should be able to

- use the MATLAB IDE to operate MATLAB interactively from within the command window;
- create and name MATLAB variables and assign them numerical values;
- invoke several built-in MATLAB mathematical functions (such as the sine, cosine, and exponential functions);
- get more information on MATLAB statements and functions using the `help` and `doc` commands;
- write a simple program that set the values of variables, calculates some quantities, and then displays the results in the command window;
- run through a program line-by-line using the MATLAB debugger in the Editor window.

1.1 Running the MATLAB IDE

MATLAB is normally operated from within the MATLAB IDE. You can launch MATLAB in the Windows environment by double-clicking on the shortcut on your desktop or by selecting it from the Start | Programs menu (Figure 1.1).

The IDE is organized into a header menu bar and several different windows. Which windows are displayed can be determined by checking or unchecking

Learning to Program with MATLAB: Building GUI Tools, Second Edition. Craig S. Lent.

Companion website: www.wiley.com/go/learningtoprogramwithmatlab2e

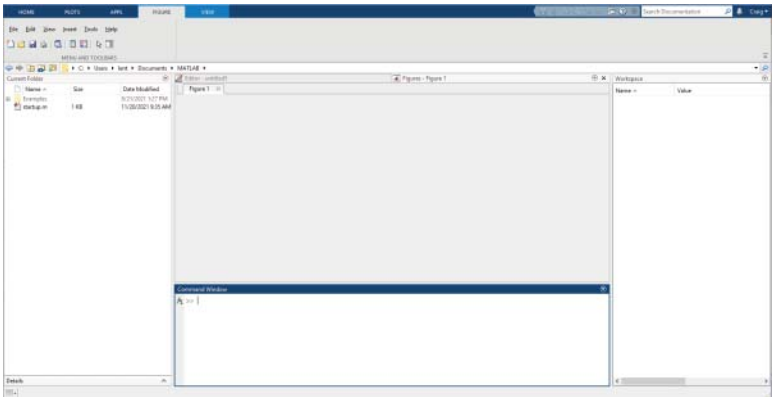


FIGURE 1.1 The MATLAB integrated development environment (IDE) with the default layout.

items in under the Desktop menu in the top menu bar. Some important windows for working with MATLAB are the following:

Command window. This is the main interactive interface to MATLAB. To issue a MATLAB command (also called a MATLAB statement), type the command at the >> prompt and press Enter.

Workspace browser. Each variable defined in the current workspace is represented here. The name, class (type), value, and other properties of the variable can be shown. Choose which properties to show using the View—Choose Columns menu from the header menu bar. A recommended set to display is: Name, Value, and Class. Double-clicking on a variable brings up the Variable Editor window. The icon representing numbers is meant to symbolize an array, i.e., a vector or a matrix. MATLAB's basic data type is the array—a number is treated as a 1×1 array.

Current Folder browser. In Windows parlance, the current folder is the same as the current directory. Without further instruction, MATLAB will save files in the current folder and look for files in the current directory. The browser displays files and other directories (folders) that reside in the current directory. Icons at the top of the browser allow the user to move up a directory (folder) level or to create a new folder. Double-clicking on a displayed folder makes it the current folder.

Editor window. The MATLAB editor is where programs are written. It doubles as part of the debugger interface, which is covered in detail later. The editor "knows" the MATLAB language and color codes language elements. There are many other convenient features to aid code-writing.

Figures window. Graphics is one of the main tools for visualizing numerical quantities. The results of executing graphics-related commands, such as those for plotting lines and surfaces, are displayed in the Figures window.

Variable Editor. The value or values held in a particular variable are displayed in a spreadsheet-like tool. This is particularly useful for arrays (matrices and vectors).

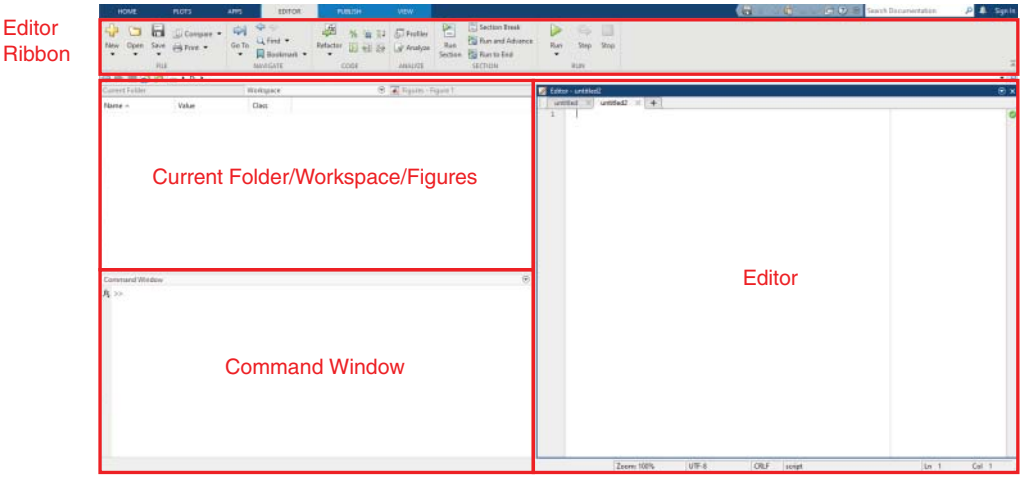


FIGURE 1.2 Recommended layout of the MATLAB IDE windows.

Manipulating windows

As usual in Windows, the currently active window is indicated by the darkening of its blue frame. Each window can be undocked using the small arrow button near the upper right-hand corner of the window. Undocked windows can be arranged on the screen using the usual Windows mouse manipulations. An undocked window can be docked again using the small arrow button (this time the arrow points downward) in the upper right-hand corner of the window.

Windows can be manipulated within the IDE by clicking and dragging the top frame of the window. Outlines of the drop position of the window appear and disappear as the mouse is moved around. This takes some practice.

More than one IDE window can share the same screen pane. Choose between active windows in a single pane by using the tabs at the top, side, or bottom of the pane.

A (strongly) recommended setup for the desktop includes three panel areas. In the upper left quadrant of the IDE, position the Workspace browser, Current Folder browser, and (optionally) the Figures window. One of these three is visible at any time, with the others being accessible by clicking the labeled tab. In the lower left, have the Command window open. The right portion is then devoted to the Editor window, where most of your programming work will take place. It really helps the development process to adopt this setup or something very like it (Figure 1.2).

1.2 MATLAB variables

A MATLAB variable is simply a place in the computer's memory that can hold information. Each variable has a specific address in the computer's memory. The address is not manipulated directly by a MATLAB program. Rather, each variable is associated with a name which is used to refer to its contents. Each variable has a *name*, such as `x`, `initialVelocity`, or `studentName`. It also has a *class* (or *type*) that specifies what kind of information is stored in the variable,

FIGURE 1.3 A schematic representation of MATLAB variables `a`, `vinit`, and `fName`. Each has a name, class (type), and a current value.

and, of course, each variable usually has a *value*, which is the information actually stored in the variable. The value may be a structured set of information, such as a matrix or a string of characters.

Numbers are stored by default in a variable class called *double*. The term originates in the FORTRAN variable type known as "double precision." Numbers in the double class take 64 bits in the computer's memory and contain about 15 digits of precision. Alphanumeric strings, such as names, are stored in variables of the *string* class. Boolean variables, which can take the value *true* or *false*, are stored in variables of the *logical* class. Logical true and false are represented by a 1 and a 0. Other variable classes will be discussed later (Figure 1.3).

Variable assignment statements

The equals sign is the MATLAB assignment statement. The command `a=5` stores the value 5 in the variable named `a`. If the variable `a` has not already been created, this command will create it, then store the value. The class of the variable (its type) is determined by the value that is to be stored. Assignment statements can be typed into the Command Window at the command prompt, a double greater-than symbol, ">>."

```
>> a=4;                    % class double
>> fname="Robert";         % class string
>> temperature=101.2;      % class double
>> isDone=true;            % class logical
```

In these examples, everything after the percent sign is a comment, information useful to the reader but ignored by MATLAB.

The assignment statement will cause MATLAB to echo the assignment to the Command Window unless it is terminated by a semicolon.

```
>> a=4.2
a =
    4.2000
>> a=5.5;
>>
```

The right-hand side of the assignment statement can be an *expression*, i.e., a combination of numbers, arithmetic operators, and functions.

```
>> a=4*7+2.2;
>> r=a+b;
>> b=sin(3.28);
>> x2=x1+4*sin(theta);
>> zInit=1+yInit/cos(a*xInit);
>> k=k+1;
```

The general form of the assignment statement is

```
<variable name>=<expression>;
```

The expression is first evaluated, and then the value is stored in the variable named on the left-hand side of the equals sign. If variables appear in the expression on the right-hand side of the equals sign, the expression is evaluated by replacing the variable names in the expression with the values of the variables *at the time the statement is executed*. Note that this does not establish an ongoing relationship between those variables.

```
>> a=5;
>> b=7;
>> c=a+b    % uses current values of a and b
c =
    12
>> a=0;
>> b=-2;
>> c
c =
    12      % kept same value despite a and b changing
```

The equals sign is used to store a result in a particular variable. The only thing permitted to the left of the equals sign is the variable name for which the assignment is to be made. Although the statement `a=4` looks like a mathematical equality, it is in fact *not* a mathematical equation. None of the following expressions is valid:

```
>> r=a=4;               % not a valid MATLAB statement
>> a+1=press-2;         % not a valid MATLAB statement
>> 4=a;                 % not a valid MATLAB statement
>> "only the lonely"="how I feel"; % not a valid MATLAB statement
```

By contrast this, which makes no sense as mathematics, is quite valid:

```
>> nr=nr+1;              % increment nr
```

Variable names

Variable names are case-sensitive and must begin with a letter. The name must be composed of letters, numbers, and underscores; do not use other punctuation symbols. Long names are permitted but very long names should be used judiciously because they increase the chances for misspellings which might go undetected. Only the first 31 characters of the variable name are significant.

`xinit`	okay
`VRightInitial`	okay
`4You2do`	not okay
`Start-up`	not okay
`vector%1`	not okay
`TargetOne`	okay

`ThisIsAVeryVeryLongVariableName`	okay
`ThisIsAVeryVeryLongVariablename`	okay, but different from previous
`x_temp`	okay

Variable workspace

The currently defined variables exist in the MATLAB workspace. [We will see later that it is possible for different parts of a program (separate functions) to have their own separate workspaces; for now there is just one workspace.] The workspace is part of the dynamic memory of the computer. Items in the workspace will vanish when the current MATLAB session is ended (i.e., when we quit MATLAB). The workspace can be saved to a file and reloaded later, although use of this feature will be rare. The workspace can be managed further using the following commands:

`clear a v g`	clears the variables `a v g` from the workspace
`clear`	clears all variables from the workspace
`who`	lists the currently defined variables
`whos`	displays a detailed list of defined variables
`save`	saves the current workspace to the file called matlab.mat
`save foobar`	saves the current workspace to the file called foobar.mat
`load`	loads variables saved in matlab.mat into the current workspace
`load foobar`	loads variables saved in foobar.mat into the current workspace

1.3 Numbers and functions

While real numbers (class double) are precise to about 15 digits, the display defaults to showing fewer digits. The command `format long` makes the display show more digits. The command `format short`, or just `format`, resets the display.

Large numbers and small numbers can be entered using scientific notation. The number 6.0221415×10^{23} can be entered as `6.0221415e23`. The number -1.602×10^{-19} can be entered as `-1.602e-19`.

Complex numbers can be entered using the special notation `5.2+2.1i`. The square root of -1 is represented in MATLAB by the predefined values of `i` and `j`, although these can be overwritten by defining a variable of that name (not recommended). MATLAB also recognizes the name `pi` as the value 3.141592653589793. This can also be overwritten by defining a variable named `pi`, an extraordinarily bad idea.

Internally MATLAB represents real numbers in normalized exponential base-2 notation. The range of numbers is roughly from as small as 10^{-308} to as large as 10^{308}.

Standard numerical operations are denoted by the usual symbols, and a very large number of functions are available. Some are shown below.

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`^`	exponentiation, e.g., `1.3^3.2` is $1.3^{3.2}$
`sin(x)`	returns the sine of x
`sind(x)`	returns the sine of x degrees
`cos(x)`	returns the cosine of x
`cosd(x)`	returns the cosine of x degrees
`tan(x)`	returns the tangent of x
`tand(x)`	returns the tangent of x degrees
`atan(x)`	returns the inverse tangent of x
`atand(x)`	returns the inverse tangent of x in degrees
`acos(x)`	returns the inverse cosine of x
`acosd(x)`	returns the inverse cosine of x in degrees
`asin(x)`	returns the inverse sine of x
`sind(x)`	returns the inverse sine of x in degrees
`exp(x)`	returns e^x
`log(x)`	returns the natural logarithm of x
`log10(x)`	returns the $\log_{10}(x)$
`sqrt(x)`	returns the square root of x
`abs(x)`	returns the absolute value of x
`round(x)`	returns the integer closest to x
`ceil(x)`	returns the smallest integer greater than or equal to x
`floor(x)`	returns the largest integer less than or equal to x
`isprime(n)`	returns true if n is prime
`factor(k)`	returns prime factors of k
`sign(x)`	returns the sign (1 or -1) of x; `sign(0)` is 0
`rand`	returns a pseudorandom number between 0 and 1
`rand(m)`	returns an $m \times m$ array of random numbers
`rand(m,n)`	returns an $m \times n$ array of random numbers

See more in the interactive help: Help| |ProductHelp|MATLAB |Functions| Mathematics

1.4 Documentation

There are many MATLAB commands and functions. To get more information on a particular command, including syntax and examples, the online facilities are accessed from the command window using the `help` and `doc` commands.