1 Basic MATLAB

SKILLS ACQUIRED

After reading this chapter, you will be able to...

- Understand how computers "think" (1.1)
- Use the MATLAB interface (1.2)
- Store code in scripts and keep them tidy (1.3; 1.4)
- Identify and address errors and warnings (1.5)
- Find help within MATLAB (1.6)

To learn computer programming, one first must learn something about how computers work. Living in the modern era, you likely interact with countless computers: in your pocket, backpack, and desk, but also integrated into buildings, infrastructure, and the Internet. The goal of this chapter is to build on the understanding of technology you already have and explain how computer programming, and MATLAB, fit in. Then, we will learn how to use the MATLAB program itself, including how to enter commands and save scripts. The chapter concludes with methods for identifying and diagnosing code issues, and how to obtain help from MATLAB.

1.1 The Design Background of MATLAB

As someone learning MATLAB, you are fortunate in that computer programming is much easier now than it has historically been. One of the first, and probably the most famous, programmable digital computers was constructed in 1945 and it took six programmers (trivia fact: all were women) to rearrange its many switches, cables, and plugs to solve a given problem. As technology became more advanced, personal computers would allow individuals to program from the comfort of their home with tools far more powerful than these early computers. MATLAB, which is managed and distributed by the company MathWorks, is one culmination of this long history. More specifically, MATLAB is descended from two very important programming languages: assembly language and Fortran. First, we will review the nature of digital information, before describing some computer architecture and reviewing these languages.

1.1.1 Digital Storage

Programming in MATLAB, like all programming, involves storing digital information, which consists of an enormous number of on-off switches whose configuration encodes all the information we

use when we interact with computers. The most basic form of digital information is the **bit**, essentially a switch like a light switch that can take the values of *one* (on) or *zero* (off). This structure is descended from on–off switches used in the oldest computers.

A single bit stores little information, but when you combine them, you can store much more. For example, you can count to arbitrarily large numbers using only ones and zeros; this "base two" counting system works just like the base 10 counting system you're used to, but the next digit is carried over when the previous digit must increase past 1, rather than past 9. For example, in binary, instead of counting 1, 2, ..., 9, 10, ..., 99, 100, ..., one would count 0, 1, 10, 11, 100, 101, ..., 111, 1000, 1001, ..., 1111, 10000, and so on. Similar representations can be made for letters; for example, the ASCII character system (subsection 7.1.1) uses 7 bits to store $2^7 = 128$ possible characters, including all numerals (0–9), English-language letters (capital and lower-case), and punctuation (commas, periods, etc.). Images (Section 8.1) are also stored in arrays of varying numbers of bits. When viewing digital files, we often see quantities of digital information referenced in bytes; a **byte** is 8 bits, such that while a bit can store only two possible values, a byte can store $2^8 = 256$ possible values.

In short, all digital information is stored in the same way: ones and zeroes. Although working with the ones and zeroes directly is not typically done in modern programming, you will be surprised how often this knowledge helps you understand programming concepts.

1.1.2 A Few Computer Components

Now that we know how digital information is stored, we can discuss how computers work with the bits they store. Figure 1.1 illustrates the main relationships between three core computer components: the **central processing unit (CPU)**, **random access memory (RAM)**, and the non-volatile storage device or **disk**. The disk and RAM store digital information in its most basic form: bits. You will often see hard drive and RAM storage quantified in some multiple of bytes; for example, a **gigabyte** is 1 billion bytes. The disk and RAM differ in their usage: the disk stores mostly static information, like text files, images, videos, and computer programs. In contrast, RAM stores the information the

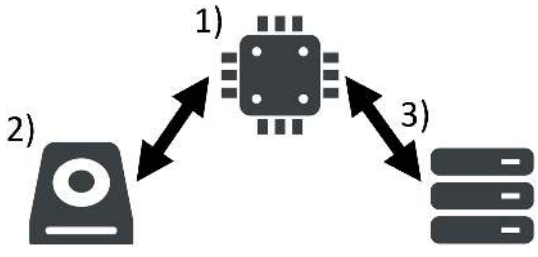

Figure 1.1 Conceptual illustration of relationships between a computer's (1) CPU, (2) disk, and (3) RAM. The CPU reads and writes data to and from the disk and RAM. RAM is used for shorter-term storage and is erased when the computer powers off, while the disk is used for longer-term storage and maintains its information when the computer powers off. Credit: Getty Images/-VICTOR-

computer is working with at the time and is nearly constantly changing as the computer runs. These differences are reflected in the technology: disks are designed to store a lot of information but cannot retrieve (read) or store (write) that information as quickly as RAM. RAM, unlike the disk, is also cleared when your computer turns off; most modern computers have a "sleep" mode that dumps the contents of RAM to the disk, powers off the computer, then restores that information upon powering back on, allowing the user to resume their work.

The third component, the CPU, performs operations by reading from disk or RAM, modifying the values, and then storing the results in disk or RAM; typically, the CPU interfaces with RAM because it can read and write faster than the disk. CPUs operate in *cycles*, where in each cycle the CPU can manipulate a number of bits equal to its capacity; at the time of this writing, most modern computers are 64-bit, allowing 64 bits of information to be manipulated on each cycle. CPU performance is measured in **hertz**, a frequency measure indicating how many cycles occur per second. For example, the base speed of the computer being used to write this book is 4.20 **gigahertz**; as with *giga*bytes, the value is multiplied by 10^9, so my computer, which is 64-bit, can execute 4.20 billion cycles, processing 64 bits of information each cycle, per second. This is extremely fast, and is representative of a modern high-performance desktop computer.

Using these components, modern computers can store and manipulate massive amounts of information at incredible speeds. Computations that would take you days, if not years, to complete by hand can be accomplished by MATLAB in an instant. When computer programs take longer than expected, keep in mind that "simple" operations like basic math and rearranging data should not take long, and if they do, this may indicate problems with your code.

1.1.3 Where MATLAB Fits In

In the words of John Backus, the creator of MATLAB:

Much of my work has come from being lazy. I didn't like writing programs, and so, when I was working on the IBM 701, writing programs for computing missile trajectories, I started work on a programming system to make it easier to write programs.[1]

Backus was referring to Fortran, an older programming language built on assembly language, an *even older* language, but the same idea applies to MATLAB at an additional level of simplification. Using MATLAB, you can write code whose function you can articulate (to some degree) in plain language, making it a so-called *higher-level* language. Programming has never been more accessible, and many of the design features of MATLAB are geared toward ease of use. If you've learned "lower-level" programming languages like C, C++, or Javascript, you'll notice MATLAB takes care of many things automatically that these languages require the programmer to do, such as defining variable types. As a result, MATLAB has much of the functionality of these "lower-level" languages, but is generally much more user-friendly.

[1] Stegmann, C. (1979, July/Aug) Pathfinder. *Think*. See https://www.softwarepreservation.org/projects/FORTRAN/paper/Backus-Think.pdf

1.2 Basics of Programming

In this section, you will do your first programming and be introduced to MATLAB code. This section is essential, and it is important to understand all aspects of each piece of code.

1.2.1 How to Use MATLAB

When you open MATLAB, you will see a screen like Figure 1.2.

You may see some extra text below the words **Command Window** (number 2 in Figure 1.2) describing your license; this is normal. If you see other text here (especially red text), and your code does not behave as it should, contact your instructor as your installation may have problems.

This window may be overwhelming at first, so let's learn the basics. The MATLAB window consists of a **taskbar** at the top and one or more **panes**, or windows-within-the-window, below. In Figure 1.2, the titles of the panes are marked as 1, 2, and 3.

1 The **Current Folder** pane displays the contents of the folder specified directly up and to the right of it. On my computer, this directory, by default, is `C:/Users/Max/Documents/MATLAB`. This pane is a file explorer, very similar to Windows Explorer on Windows, Finder on Mac, or a cloud-based file browser. Double-clicking a folder enters it, double-clicking a file attempts to open it with MATLAB, and the icons at the top of the pane navigate to nearby folders. This **Current Folder** pane is surprisingly important (Section 9.1), so do not forget about it.
2 The **Command Window** pane is where MATLAB code goes to execute, or "run". Here, you can enter MATLAB commands, then press Enter to run them, and view their output. The primary way to interact with MATLAB is through commands, so we'll be here a lot.

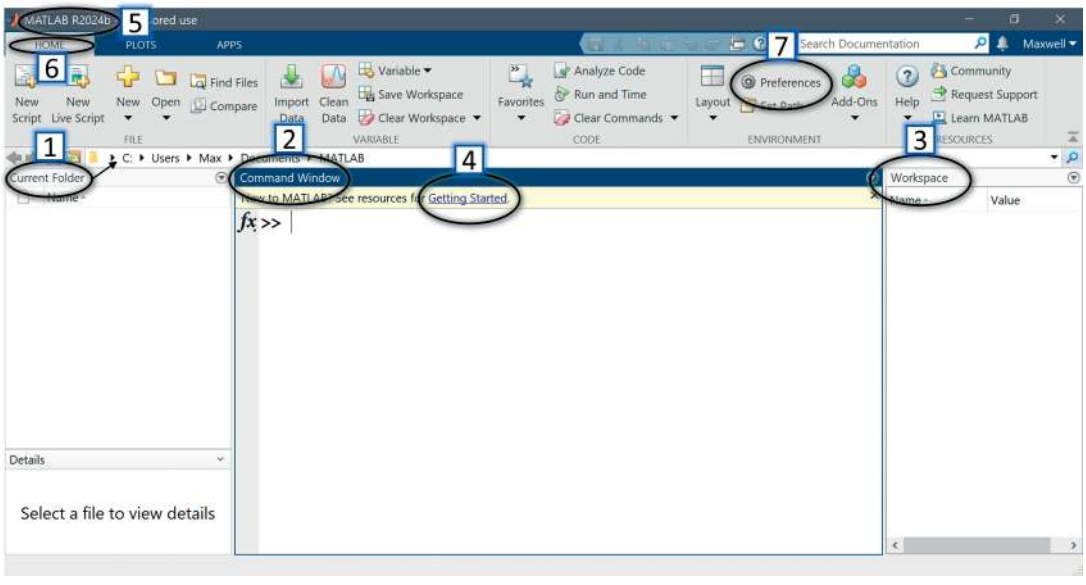

Figure 1.2 MATLAB when opened for the first time. MATLAB screenshot.

3 The **Workspace** lists the variables MATLAB currently stores in RAM. We will learn more about variables shortly, and we will revisit this pane as it becomes populated with information.

A few other things are worth noting, each of which is labeled in Figure 1.2.

4 If you recently installed MATLAB, you will see a link to **Getting Started** at the top of the **Command Window**. Click it, and a documentation window will open with links to help you get started with MATLAB. These links are very helpful, and are worth exploring at some point, but if you are overwhelmed already, simply read on and come back later. You can always also get to this resource by typing `doc get started` in the **Command Window**, pressing Enter, and clicking the first link.
5 MATLAB's version number is listed in the top-right. MATLAB releases a new version a couple of times a year; for example here "R2024b" was released in the second half of 2024. You'll likely see a later version here; that's fine, most of MATLAB doesn't change much across versions, but you may see slight differences in output or (more rarely) syntax.
6 The **Toolstrip** at the top of MATLAB contains helpful tools for working with MATLAB. Some of them we will never touch; for all icons, hover over a button with your cursor for a brief description. Others, like the **New Script** button, will be used often. Hovering over a button also reveals the hotkey for that button, a combination of keyboard keys used to perform the same action via the keyboard. For example, on Windows, the **New Script** button is activated with Ctrl+N. The **Toolstrip** is divided into several **tabs**, or alternate versions of the **Toolstrip** containing different icons and tools. In Figure 1.2, the **Home** tab is selected (number 6). Clicking on another tab will reveal different icons, although some icons are shared across tabs. Sometimes tabs will appear and disappear depending on what you are doing, and we will see an example when we start creating and editing scripts in Section 1.3.
7 The **Preferences** icon, which can be accessed from the **HOME** tab, allows you to change many things about MATLAB, including some cosmetic features. For example, within Preferences, **Fonts** changes the fonts used by MATLAB in the **Command Window** and elsewhere. Some preferences change MATLAB in ways you may not expect; if this happens, you can always restore default preferences by typing `doc preferences` into the **Command Window**, pressing Enter, and following the presented instructions.

Now that we know our way around MATLAB, let's start using it. Ultimately, the best way to learn to program is to program, and we are now ready to write our first code.

1.2.2 MATLAB as a Calculator with Memory

Nearly all modern programming languages, from assembly language to MATLAB, can be described in terms of their **commands**, which differ from program to program. A command is an order or instruction for what the computer is supposed to do. See **Example** 1.1 for your first MATLAB command.

Example 1.1 Write code to compute the sum of 1 and 1.

Solution

```
1+1
```

Explanation

As a MATLAB command, `1+1` instructs MATLAB to perform this calculation and return the result (2). If you type this into the **Command Window** and press Enter, you will see the following output:

```
ans =
     2
```

We just used MATLAB as a calculator. MATLAB is often useful for this purpose alone.

What if you want to store the result of a calculation? In algebra, we often see values represented by **variables**, proxies for numbers whose values can change over time. Programming languages use variables in much the same way: storing a value in a variable is called **assignment**, and its companion operation, called **reference**, involves retrieving a value stored in a variable. In MATLAB, the equals sign, (=), is used for assignment (**Example** 1.2).

Example 1.2 Store the value 1 in a variable named `x`.

Solution

```
x=1
```

Explanation

The syntax for (=) is rather specific, and order matters: the value on the right-hand side is stored in the variable whose name is given on the left-hand side. When you run this command, you will see the following output:

```
x =
     1
```

This output structure, a variable followed by the (=) sign followed by a value, was also produced in the `1 + 1` example; there, we did not use assignment (i.e., did not include (=)) and therefore, by default, MATLAB stores the result in a variable called `ans` ("answer"). MATLAB operations that produce a result (not all do) will perform assignment whether you want them to or not; when you do not specify the variable name, the result is stored in `ans`.

Reference is easy: simply use the variable in a subsequent command (**Example** 1.3).

Example 1.3 Add `x` to itself twice, storing the result in the default variable `ans`.

Solution

```
x + x + x
```

Explanation

As in algebra, MATLAB will substitute the value of `x` each time it is used, so `x + x + x` is replaced by `1 + 1 + 1` within MATLAB, yielding `ans = 3`.

The "vari" in "variable" indicates that, as in algebra, variables can change over time. The command above changes the value of `ans` from 2 to 3; if you've only run the commands I've listed here, in order, running `ans + ans` before the previous `x + x + x` command would have yielded 4, but after running `x + x + x`, the same command will yield 6. Similarly, if we first run

```
x = 5
```

then `x + x + x` yields 15, not 3. We have stored the results of calculations in `ans`, but we can store them in variables in the same way. For example, after running `x = 5`, the command

```
y = x * 3
```

stores the value 15 in `y`, where the asterisk (`*`) is used to perform multiplication in MATLAB.

Just as (`+`) performs addition and (`*`) performs multiplication, (`-`) performs subtraction, (`/`) performs division, and (`^`) performs exponentiation. These symbols are called **operators**, special characters that perform some action in code. The terms on either side, upon which the operators operate, are called **operands**. For individual numbers, these operators work just like in math class: order of operations applies, such that exponentiation is performed before multiplication and division, which are performed before addition and subtraction, and parentheses can be used to customize order of operations; see subsection 4.2.3 for more details. When we get to arrays (Chapter 2), we'll see some different uses of these operators.

You can now use MATLAB as a calculator with memory. In some sense, this is *all* MATLAB is, but it is an extremely flexible calculator with lots of features that make it much more useful and powerful than your typical calculator. One important restriction is that variable names must be valid for MATLAB; see Box 1.1 for more information on MATLAB variable name specification.

1.2.3 The Workspace Pane

One useful MATLAB feature is the **Workspace** pane to the right of the **Command Window** (numbered as 3 in Figure 1.2). If you have entered all commands that were printed on separate lines from the rest of the text, and nothing else, your **Workspace** pane will look like Figure 1.3.

Variables (`ans`, `x`, and `y`) are presented with their values. As the variables we store change, the information displayed here will also change.

Box 1.1 HOW TO: Construct proper variable names in MATLAB

MATLAB has specific requirements for valid variable names. In particular:

- Valid variable names must start with a lower-case letter (a–z) or upper-case letter (A–Z). Case (capital versus lower-case) matters; `VARIABLE`, `Variable`, `variable`, and `VaRiAbLe`, and so on are all different and will be treated as such by MATLAB.
- All other characters in a variable name must be letters, digits 0–9, or underscores (_).
- MATLAB has a maximum variable length; the `namelengthmax` command will reveal the number of characters (letters, digits, underscores) that can comprise a variable name.
- Special MATLAB keywords, such as `if`, `else`, and `end`, cannot be used as variable names. The `iskeyword` command provides a full list. These keywords appear in blue text in the **Command Window** and **Editor** and many are used in all but the simplest programs, so you will quickly learn to recognize them.

Variable names should generally be descriptive and complete, such that when you or someone else reads your code later, it is obvious what variable names refer to.

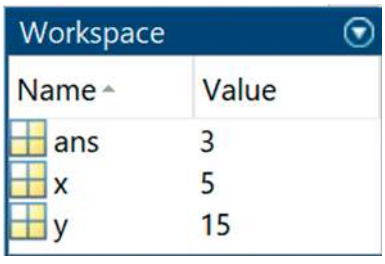

Figure 1.3 The **Workspace** pane after adding some variables. MATLAB screenshot.

1.3 Scripts

Thus far, we have only typed commands into the **Command Window** and viewed the output. This is a valid way to use MATLAB, and will remain useful for quick calculations and testing code as you write it, but it has some serious shortcomings. First, if you close MATLAB and open it again, your **Workspace** will disappear (specifically, your RAM will be cleared of all variables) and you will need to run your previous commands again to get them back. You can use the up and down arrow keys on your keyboard to scroll through previous commands, but this is tedious and you may need to scroll through many commands to reach the code you are seeking. The answer to these issues is to store your code in a **script**, which is a plain text file (no formatting) containing a sequence of MATLAB commands. Let's create one.

1.3.1 Creating a Script

To create a script, press the **New Script** button in MATLAB; if you are in the **HOME** tab, it should be in the top left of the window. After you do, MATLAB appearance will change in the following ways (Figure 1.4).

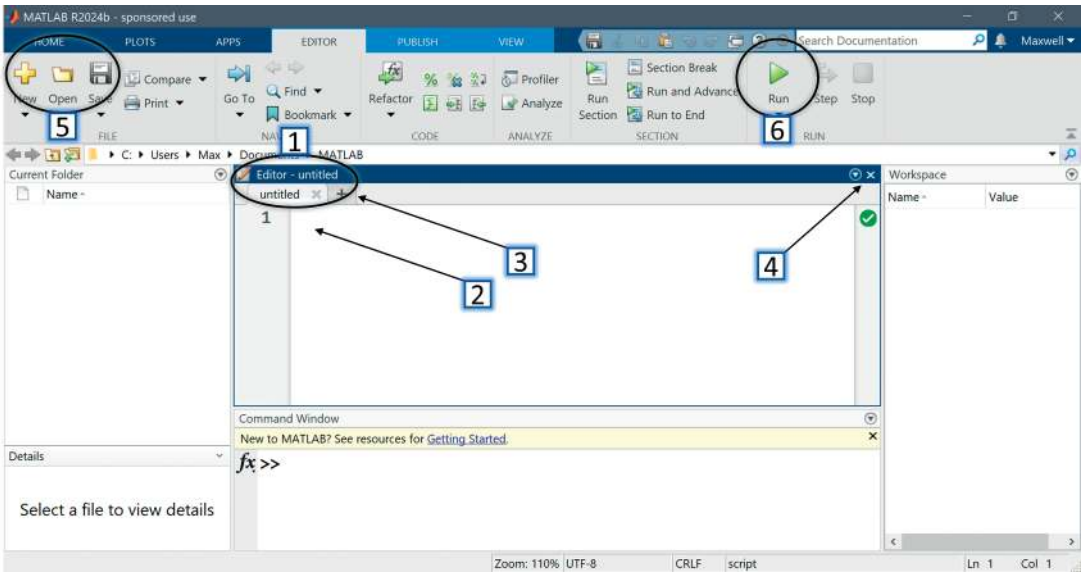

Figure 1.4 The Editor when first opened in MATLAB. MATLAB screenshot.

A new **Editor** pane will appear, this is the **Editor** and you will be using it to write scripts. By default, opening a new script splits the **Command Window**, such that the top part is now the **Editor** and the bottom part remains as-is. You should be aware of a few important components.

1. The label for the **Editor** can be dragged to rearrange panes within the MATLAB window. You can also see the title of the current script (`untitled`) and an **X** symbol that closes the script.
2. This blank box is where you type your code. Lines are numbered, note the `1` on the left-hand side of the screen for the first line of code, and more numbers will appear as you write your script.
3. Click this **+** symbol to open a new script. You can have as many scripts open at a time as you like, although it might get hard to navigate among many, so try to keep as few open as you need.
4. Lastly, to close the **Editor**, click the **X** in the top-right corner.

Second, a new tab, also with the title **EDITOR**, will appear in the **Taskbar**. You can hover over each component to see what it does and its associated hotkeys. I'll draw attention to a few.

5. In the top left, you can see easy-to-access commands to make a new script, open a script, or save the script you are currently working with.
6. **Run** runs all commands in a script, starting with the first command and then executing subsequent commands, line by line, until the end. Two hotkeys are helpful:
   - on Windows, pressing F5, or on Mac, pressing F5 or Ctrl+Alt+R, is equivalent to pressing the **Run** button, and will run all lines in the script that is currently open;

IMPORTANT! The **Save** button is perhaps the most important button in MATLAB. Use it often, lest you lose your work. For Windows, the hotkey is Ctrl+S while for Mac the hotkey is Command+S. Try to train use of this hotkey to be a reflex, like pressing the spacebar in between words. Develop this habit early, before you fail to save a major script and need to redo hours of work. You have been warned.

- if you want to run only a specific piece of code, you can select that code by clicking and dragging your cursor over it then pressing F9 on Windows or Shift+F7 on Mac.

On laptops and some desktop keyboards, you may need to press the Function (sometimes abbreviated to Fn) key to press the F buttons (F5, F7), as many keyboards also use these keys for things like playing media, increasing or decreasing volume, and so on.

Once your script is open, you can type commands into it, then run all commands in the script with a single button press. As an example, try typing some of the first commands we used:

```
x = 1
x + x + x
```

Click the green **Run** button at the top of the screen (Figure 1.4, number 6). MATLAB will give you a little trouble at first; a script needs to be saved to the disk to be executed, so MATLAB will ask you to do so, but these files are usually very small. If you haven't yet, it might be useful to make a folder to store your MATLAB scripts. The default file name is `untitled`; that will do for now, so go ahead and save the script. It will have the extension `.m` identifying it as a MATLAB script, so the full file name is `untitled.m`. MATLAB might also give a warning message about a path; either option gets the script running. See Section 9.1 for more information on paths.

Now that the script is saved, we can run it. Press **Run** and the commands in the script will execute, one line at a time, starting from the top and ending at the bottom. In Chapter 5, we will change the order in which commands are run, but generally commands will run from top to bottom.

Scripts serve two purposes: saving your code for use later, and producing a clean, predictable sequence of commands. This sequence of commands is simple, but as the things we do with MATLAB become more complex, it will become increasingly important to maintain control over the sequence of commands that are run. Do not worry, we will build up to this complexity over time.

1.4 Script Style

Programming **style**, non-functional aspects of code that affect how it displays in the **Editor**, is not the focus of this textbook. This is intentional; students, especially those new to programming, tend to perform better when evaluated on what their code *does*, rather than how it *looks*. In general, you should tailor your script style to the demands of the task; a simple script to count calories in your lunch requires a different degree of perfection than production code for self-driving cars. However, there are a few aspects of style that provide immediate and direct value.