Chapter 1

Introduction

The MATLAB software package is used for computation in engineering, science, and applied mathematics. It offers a powerful programming language, excellent graphics, and a large standard library.

The focus in MATLAB is on computation, not mathematics: symbolic expressions and manipulations are not possible, except through the optional Symbolic Toolbox, which is not covered in this book. All variables must have values, and all results are numerical and potentially inexact, thanks to the rounding errors inherent in computer arithmetic. The emphasis on numerics is typical for most work in scientific computation.

Compared to other numerically oriented languages, such as C++ and Fortran, MATLAB is usually found to be much easier to use. However, its execution speed can be slower. This gap is not always as dramatic as popular lore has it, and it can often be narrowed or closed with good MATLAB programming. Moreover, one can link other types of code into MATLAB, or vice versa, and MATLAB has some optional support for parallel computing. Still, MATLAB is usually not the tool of choice for high-performance computing.

Whatever you think of these or other limitations of MATLAB, they have not held back its popularity: a recent search for “matlab” on the books section of Amazon.com turned up 8,543 results! Rapid code development and interaction with data often trump execution speed, and the integrated graphics and expert routines that come with MATLAB can be decisively helpful. Even for speed-hungry users, MATLAB can be a valuable environment in which to explore and fine-tune algorithms before creating production code in another environment.

Successful computing languages and environments reflect a distinctive set of values. In MATLAB, those values include an emphasis on experimentation and interaction with data and algorithms; syntax that is compact, friendly, and interactive (rather than tightly constrained and verbose); a kitchen-sink mentality for providing functionality; and a predilection for vectors, matrices, and arrays.

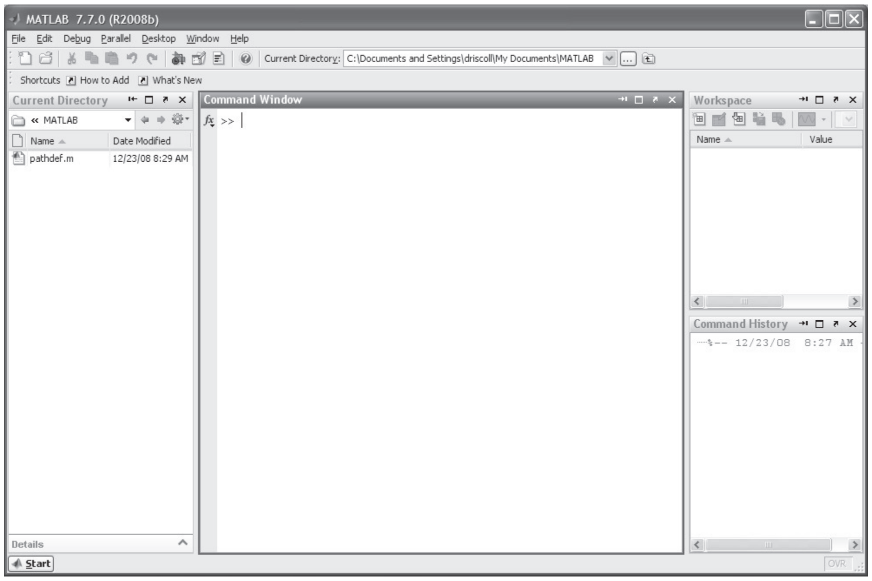
MATLAB 7.7.0 (R2008b)
File Edit Debug Parallel Desktop Window Help
Current Directory: C:\Documents and Settings\driscoll\My Documents\MATLAB
Shortcuts How to Add What's New
Current Directory
« MATLAB
Name Date Modified
pathdef.m 12/23/08 8:29 AM
Details
Command Window
>>
Workspace
Name Value
Command History
--%-- 12/23/08 8:27 AM
Start
OVR

Figure 1.1. *Initial MATLAB desktop window. (Microsoft® Windows® XP version shown.)*

1.1 The fifty-cent tour

When you start MATLAB, you get a multipaneled desktop, as seen in Figure 1.1. The layout and behavior of the desktop and its components are highly customizable. The component that is the heart of MATLAB is called the **Command Window**, located in the middle by default. Here you can give MATLAB commands typed at the **prompt**, shown as >>. Unlike Fortran and other compiled computer languages, MATLAB is an interpreted environment—you give a command, and MATLAB tries to execute it right away, then awaits another.

At the left you can see the **Current Directory** window. In general MATLAB is aware only of files in the current directory (folder) and on the list known as its **path**, which can be customized. Commands for working with the directory and path include `cd`, `what`, `which`, `addpath`, and `editpath` (or you can choose Set path ... from the File menu). You can add files to a directory on the path and thereby add commands to MATLAB; we will return to this subject in Chapter 3.

At the top right is the **Workspace** window. The Workspace shows you what variable names are currently defined and some information about their contents. At start-up it is, naturally, empty. This represents another break from compiled environments: variables created in the workspace persist for you to examine and modify, even after code execution stops.

Below the Workspace window is the **Command History** window. As you enter commands, they are recorded here. This record persists across different MATLAB sessions, and commands or blocks of commands can be copied from here or saved to files. Thus the Command History is very useful if you realize belatedly that you need to save some or all of what you have been doing interactively.

As you explore MATLAB, you may encounter some **toolboxes**. These are individually packaged sets of capabilities that provide in-depth expertise on particular subject areas. There is no need to issue a command to load them—once installed, they appear on the path and are available automatically.

1.2 Graphical versus command-line usage

Originally, MATLAB was entirely a command-line environment, and it retains a bias in that direction.[1] But it is possible to access a great deal of the functionality from graphical widgets such as menus, buttons, and so on. These interfaces are especially useful to beginners, because they lay out the available choices clearly. In particular, take time to right-click (or Control-click on a Mac®) on various objects to see what you might be able to do with them.

As a rule, graphical interfaces can be more natural for certain types of interactive work, such as annotating a single graph, whereas typed commands remain better for complex, precise, repeated, or reproducible tasks. Much of the time, you can choose whichever mode of operation suits you. For instance, you can write a function that customizes any figure's appearance, but you can also save aspects of a current figure's style as a template just by pointing and clicking. Moreover, you can create your own graphical interfaces and even distribute them with your code as a package for non-MATLAB users. In the end, an advanced MATLAB user should be able to exploit both modes of operation. That said, the focus of this book is on typed commands. In many, perhaps most, cases these have graphical interface equivalents, even if I don't explicitly point them out.

1.3 Help

MATLAB is huge. Nobody can tell you everything that you personally will need to know; nor could you remember it all anyway. It is essential that you become familiar with the online help. There are two levels of help:

- If you need quick help on the syntax of a command, type `help` in the command window. For example, `help plot` shows directly in the Command Window all the ways in which you can use the `plot` command. Typing `help` by itself gives you a list of categories that themselves yield lists of commands.

- Typing `doc` followed by a command name brings up more extensive help in a separate window. For example, `doc plot` is better formatted and more informative than `help plot`. In the left panel one sees a hierarchical, browsable display of all the online documentation. Typing `doc` alone or

[1]You can still run MATLAB entirely in this mode by starting it with the `-nojvm` option, which can be convenient when connecting to a remote server, for example.

selecting Help from the drop-down menus brings up the window at a root homepage.

Starting in MATLAB 7.7, the Command Window includes a **function browser** whose icon is *fx*, sitting next to the prompt. Clicking on it yields a searchable, hierarchical listing of available functions, with help available on one more click. Merely pausing after typing the name of a function and its opening parenthesis brings up a context-sensitive box with syntactic suggestions, as well.

The *Getting Started with MATLAB* manual is a good place to get a more leisurely and thorough introduction than the one to follow here. Depending on your installation, the documentation may be available in PDF form for printing and offline reading, or you can find it on the Web at: www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf

Complementary to the online help is MATLAB Central, found on the Web at www.mathworks.com/matlabcentral. It includes a discussion forum and the File Exchange, which consists of code contributions by MATLAB users and friends. It's a good place to check when you suspect that you may be trying to reinvent the wheel.

1.4 Basic commands and syntax

If you type in a valid expression and press Enter, MATLAB will immediately execute it and return the result, just like a calculator.

```
>> 2+2
ans =
     4

>> (6.02e23)^2
ans =
  3.6240e+047

>> sin(pi/2)
ans =
     1

>> 1/0

ans =
   Inf

>> exp(i*pi)
ans =
  -1.0000 + 0.0000i
```

Notice some of the special expressions here: pi for π, Inf for ∞, and i for $\sqrt{-1}$. You may also use j for the imaginary unit. (Both names can be reassigned, however, and are commonly used as integers. It's safer always to use 1i or 1j to refer to the imaginary unit.) Another special value is NaN, which stands for **not a number**. NaN is used to express an undefined value. For example,

```
>> Inf/Inf
ans =
   NaN
```

NaN can be tricky to use: two NaN values are unequal by definition, for instance.

You can assign values to variables with alphanumeric names:

```
>> x = sqrt(3)
x =
    1.7321

>> days_since_birth = floor(now) - datenum(1969,05,06)
days_since_birth =
       12810

>> 3*z
??? Undefined function or variable 'z'.
```

Observe that *variables must have values before they can be used*. When an expression returns a single result that is not assigned to a variable, this result is assigned to ans, which can then be used like any other variable:

```
>> atan(x)
ans =
    1.0472

>> pi/ans
ans =
     3
```

In floating-point arithmetic, you should not expect "equal" values to have a difference of exactly zero. The built-in number eps, often called **unit roundoff** or **machine precision** in numerical analysis, bounds the maximum relative error in representing real numbers and doing arithmetic on your particular machine.[2]

[2]Like all other names, eps can be reassigned, but doing so has no effect on the precision.

```
>> eps
ans =
   2.2204e-16

>> exp(log(10)) - 10
ans =
   1.7764e-15

>> ans/10
ans =
   1.7764e-16
```

Here are a few other demonstration statements.

```
>> % Anything after a % sign is a comment.
>> x = rand(100,100); % ; to suppress output
>> s = 'Hello world'; % single quotes enclose a string
>> t = 1 + 2 + 3 + ...
4 + 5 + 6             % ... continues a line

t =
    21
```

Once variables have been defined, they exist in the workspace. You can see what's in the workspace from its desktop window, or by typing `who` or `whos`:

```
>> who

Your variables are:

ans  s    t    x
```

1.5 Saving and loading work

If you enter `save myfile`, all defined variables will be saved to a file called myfile.mat in the current directory. This file format is particular to MATLAB. You can also select a subset of variables to be saved by typing their names after the file name. If you later enter `load myfile`, the saved variables are returned to the workspace, overwriting any presently defined values assigned to the same names.

If you highlight commands in the Command History window, right-click, and select Create M-File you can save the typed commands to a text file. This can be very helpful for recreating what you have done. Another way to save both input and output is to use the `diary` command with a file name. This causes all subsequent commands and results to be recorded in a text file on disk. A more sophisticated version of the same idea is called **publishing**, which is discussed in section 3.1 on page 28.

MATLAB is also capable of loading and saving other common file formats, such as formatted text files, spreadsheet files, and common graphics and video files. To load such a file, it's often easiest to double-click it in the Current Directory window, and follow the resulting prompts.

1.6 Things about MATLAB that are very nice to know, but which often do not come to the attention of beginners

Don't worry if everything in this list doesn't make sense yet—come back to it again later.

(a) Use the up-arrow key to cycle through previous commands. If you type specific characters first, only commands matching the typed characters will be recalled.

(b) If a computation is taking too long, interrupt it by pressing Ctrl-C (after making sure the Command Window is active in the operating system).

(c) Even when MATLAB displays only 4–5 digits of a result, it's storing about 15 significant digits. (You can see them all by typing `format long`.) By copying or retyping a displayed result, you throw away a lot of information. Wait until the end of the calculation to round off results.

(d) MATLAB has great debugging tools. Run your code step by step to uncover errors. Run someone else's code step by step to understand it thoroughly.

(e) The M-Lint code checker makes some good suggestions.

(f) The previous two items alone are sufficient reasons to use the built-in MATLAB Editor for writing code. Open it by entering `edit`.

(g) If the execution of your code is too slow to suit, use the Profiler to find the slowest steps. Colleagues will be amazed at how you always home in on the bottlenecks.

(h) If you want to share code or use it on a different machine, first use the `depfun` command on it to find the files it depends on. Or, check out the section of online help on "directory reports."

(i) Don't use a screen or window capture function to paste figures into a document or presentation. The results look cheesy and amateurish. Instead, have a look at section 5.5.

(j) After you have properly exported a figure to a graphics file, save that figure again in the native FIG format. You may want to make changes to it someday.

(k) Plot legends are helpful and can be generated automatically. Better still are curves with labels right next to them—also very easy to create.

(l) Anonymous functions are really powerful. Trust me on this, and go learn about them, beginning in section 4.1.

Exercises

1.1. Evaluate the following mathematical expressions in MATLAB:

(a) $\tanh(e)$	(b) $\log_{10}(2)$	(c) $\log_2(10)$
(d) $\left\|\sin^{-1}\left(-\frac{1}{2}\right)\right\|$	(e) 123456 mod 789	(f) $\operatorname{Arg}(1+i\sqrt{2})$

1.2. MATLAB ships with some interesting data. For example, try the following:

```
load usapolygon, plot(uslon,uslat)
```

Use `who` or the Workspace browser to find out where the data comes from.

1.3. What is the name of the built-in function that MATLAB uses to

1. compute a Bessel function of the second kind?
2. test the primality of an integer?
3. multiply two polynomials together?
4. plot a vector field?
5. report the current date and time?

1.4. Find a function on the MATLAB Central File Exchange that enables you to delete the most recently created graphics object from the command line.

1.5. MATLAB ships with some built-in capability for working with images (especially if you have the Image Toolbox available). After all, an image is usually represented as either one array or three arrays of pixel intensity values. Use the commands

```
>> load durer
>> image(X)
>> colormap(map)
```

to see a reproduction of a mathematically themed work of art. Then use the online documentation to find out more about the matrix that appears in this artwork.

About the MATLAB Treasure Hunt on the next page: I load a briefcase with a reward and secure it using a three-digit luggage lock. Teams or individuals work to find the last answer in the hunt and try it as the combination of the lock. The first team to open the lock keeps the bounty.

A MATLAB Treasure Hunt

Follow the directions. You may use only MATLAB and its local online help.

Find the largest prime factor of 20830123: $\alpha =$ ________

Find the complete elliptic integral of the first kind, $K(1-1/\alpha^2)$, rounded to three significant digits: $\beta =$ ________

Find the remainder after the largest possible positive 32-bit integer in MATLAB is divided by 100β: $\gamma =$ ________

Find the maximum element in a $\gamma \times \gamma$ symmetric Clement matrix, rounded to the nearest integer: $\delta =$ ________

Find the number of minutes that elapsed between January 20, 1961 at 12:51 PM, and July 16, 1969 at 9:32 AM. Divide by 100δ, and round to the nearest integer: $\epsilon =$ ________
