

2.1 *MATLAB* Tutorial 1

Introduction to Computational Science: Modeling and Simulation for the Sciences

Angela B. Shiflet and George W. Shiflet

Wofford College

© 2006 by Princeton University Press

Download

Download the file *MATLABTutorial1.m* from the textbook's website. The section on Script M-Files describes how to work with this file.

Introduction

This first tutorial on *MATLAB*® gives an introduction to the system and prepares you to use *MATLAB* to complete several projects in the first few chapters. The tutorial introduces the following functions and concepts: Getting started; evaluation; saving; numbers; arithmetic operators; built-in functions, such as *log*, *sin*, and *exp*; variables; assignments; user-defined functions; online documentation; *disp*; looping with *for*; *plot*; differentiation; solving differential equations; and optional material on integration. The module gives the input and output of computational examples along with Quick Review Questions for you to do with *MATLAB*.

We recommend that you work through the introduction with a copy of *MATLAB*, answering all Quick Review Questions. *MATLAB* is available from MathWorks, Inc. (<http://www.mathworks.com/>).

Getting Started

MATLAB® is a powerful system that can plot graphs and perform a large variety of calculations with numbers, particularly with **matrices** or rectangular arrays of numbers. By including the Symbolic Math Toolboxes, we also can perform calculations with symbols.

If it is not open, double-click the *MATLAB* folder to open it and double-click the *MATLAB* icon to start and view a *Desktop*, which has a *Command Window* on the right. For the following material, at a **prompt** of **>>**, type each example; and evaluate the statement in the command window. To evaluate a command, press **<ENTER>** or **<RETURN>**. In this document, input is in red, and the resulting output is in blue. We start by evaluating 12-factorial, 12!, which is the product of the positive integers from 1 through 12, using the built-in *factorial* function as *factorial*(12). After pressing **<ENTER>** or **<RETURN>** for the first computation, *MATLAB* returns the result, as following prompt (**>>**), input (in red), and answer (in blue) indicate:

```
>> factorial(12)
```

```
ans =
```

```
479001600
```

For 25!, the following output appears in exponential notation with e+25 indicating multiplication by 10^{25} , so that the value is 1.5511×10^{25} :

```
>> factorial(25)

ans =

1.5511e+25
```

Quick Review Question 1 Evaluate 100-factorial.

Additional Features

To have the command window be larger, from the *Desktop* menu, we select *Desktop Layout* and then *Command Window Only*. This operation removes the *Command History* and *Current Directory* windows from the desktop. To clear the command window, from the *Edit* menu, we select *Clear Command Window*.

Frequently, we need a **comment** or explanation of material, which we do not want *MATLAB* to execute. We begin a comment line with a **percent (%)** sign, such as follows:

```
% MATLAB does not execute this line.
```

Quick Review Question 2 Have the *MATLAB Desktop* contain only the *Command Window*, and clear this window. Type comment lines with your name and "Tutorial 1."

We can select text and **cut**, **copy**, or **paste** it using items from the *Edit menu* or the short cuts indicated on that menu. Moreover, pressing the **up arrow** (\uparrow) recalls the last statement, which then we can edit before pressing <ENTER> or <RETURN>. Continuing to press the up arrow recalls even earlier statements.

Sometimes, we change our mind about executing a command. Instead of pressing the <DELETE> key a number of times, we can press <ESC> to clear the command line.

Quick Review Question 3

- a. Preface this Quick Review Question with a comment that has "QRQ" and the question number, such as follows:

```
% QRQ 3
```

- b. Press the up arrow to recall the command to evaluate 100!. Select 100, type 38, and execute the revised command, which evaluates 38!.

- c. Recall the command from Part b, but do not execute it. With one keystroke, delete the command. In a comment, indicate the key you pressed to remove the command.

To print, select **Print** from the *File* menu or use the indicated short cut.

To quit *MATLAB*, choose **Quit** from the *File* menu or use the indicated short cut.

Script M-Files

We can create, edit, save, and recall our work in a **script M-file**. Start such a file by selecting from the **File menu, New**, and then **M-File**. Type the following opening comments, including your name and date:

```
% Tutorial1.m
% Commands for the Quick Review Questions in MATLAB Tutorial 1
% my name and date
```

Save the file by selecting from the *File* menu **Save As...** and entering a file name to which *MATLAB* appends the extension **.m**. Make sure you save the file to your disk. After subsequent work, save by selecting **Save** from the *File* menu or using the indicated shortcut. **SAVE OFTEN**, particularly before you print, do a long calculation, or display a graph.

To execute the entire M-file, from the **Debug** menu, select **Run** or use the indicated shortcut. Results appear in the *Command Window*. In your M-file, enter 25-factorial (25!), execute the entire file, and observe the result in the *Command Window*. Instead of executing the entire file, we can select one or more commands and from the **Text** menu select **Evaluate Selection** or use the indicated shortcut. Select and execute the command for 25-factorial. Close the file. We can open a closed file for execution and editing.

Quick Review Question 4

- a. Open the file *MATLABTutorial1.m*, which contains the quick review questions for this tutorial.
- b. From the *Command Window*, copy your answers from Quick Review Questions 1, 2, and 3b into the appropriate places.
- c. Execute the file *MATLABTutorial1.m*. In a comment, indicate the shortcut to execute the entire file.
- d. In a comment, give the shortcut to execute selected command(s).
- e. Save your new *MATLAB* file to your disk. Record all your answers to Quick Review Questions in *MATLABTutorial1.m*.

Numbers and Arithmetic Operations

Scalar multiplication is indicated by an **asterisk (*)**; while the addition, subtraction, and division operators are **+**, **-**, and **/**, respectively. An expression can be raised to a power using a **caret (^)**, as follows:

```
3 * (5/8 - 1.25)^2
```

Quick Review Question 5 Have a comment with "QRQ 5." Add the fractions (not decimal numbers) one-half and three-fourths. *MATLAB* returns a decimal expansion.

MATLAB has numerous built-in functions, such as *sin*, and built-in constants, such as *pi* representing π . Functions use parentheses around the argument(s), as in the following expression to compute $5\sin(\pi/3)$:

```
5 * sin(pi/3)
```

Quick Review Question 6 $\log_{10}(x)$ is the common logarithm of x , or logarithm to the base 10. Evaluate the common logarithm of 23.4, recording your answer in *MATLABTutorial1.m*.

Quick Review Question 7 $\log(x)$ is the natural logarithm of x , usually written as $\ln(x)$ in mathematical notation. Evaluate the sine of the natural logarithm of 23.4.

Quick Review Question 8 e^x is *exp(x)* in *MATLAB*. Evaluate the number e^2 .

Variables and Assignments

We can employ **variables** to store values for future use. We use the same rules for naming variables as we do for other symbols, such as user-defined functions, as follows:

1. Symbols must begin with a letter of the alphabet.
2. Any combination of letters, digits, and underscores or nothing at all can follow.
3. *MATLAB* is **case sensitive**, regarding uppercase and lowercase letters as different. Therefore, the names *Exp*, *exp*, and *EXP* are three separate symbols in *MATLAB*.

By convention, we begin user-defined names with a lowercase letter. A compound name, such as *rateOfChange*, has the first letter of each subsequent word capitalized.

We can **assign** a value of an expression to a variable using the following format with the variable receiving the value always appearing on the left of an **equal sign (=)**:

variable = expression

For example, the following assignment statement gives the value 5 to the variable *var*:

```
var = 5
```

MATLAB calculates the value of the expression on the right, such as 5, and then assigns the value to the variable on the left, such as *var*. If we subsequently use *var*, *MATLAB* replaces *var* with its value.

When *MATLAB* returns a value that is not the result of an assignment statement, the system assigns the value to variable *ans*. We can use this variable in subsequent expressions until *MATLAB* gives a new value to *ans*.

Quick Review Question 9

- Assign 4.8 to the variable *time* and execute the statement.
- Type *time* and execute.
- Type *time* + 3 and execute.
- Type *time* and execute. Note that execution of Part c did not change the value of *time*.
- Type *ans* and execute. Note that the result of Part c is returned.
- Evaluate *ans*³.

If we use a variable before it has a value, *MATLAB* returns an error message, such as "??? Undefined function or variable 'r'." If the Symbolic Math Toolboxes are available, to create a symbolic variable that displays as *r*, we employ the *sym* or *syms* function in one of the following commands:

```
r = sym('r')
syms r
```

This process also clears any value for a variable. To clear several variables, we can separate the variables with blanks (*not* commas) in the *syms* statement, such as follows:

```
syms r s t
```

Quick Review Question 10

- Type *vel* and execute.
- Make *vel* a symbolic variable.
- Type *vel* and execute.
- Type an expression for $(7 + vel / 3)^2$ and execute.

The assignment statement

```
units = 12 * numberOfDozens
```

is not an algebraic formula. An illustration of the difference between an algebraic equality and a *MATLAB* assignment is the following program segment:

```
counter = 5
counter = counter + 1
```

The last statement does **not** say that *counter* is one more than itself. When *MATLAB* encounters the second statement, it looks up the present value of *counter* (5), evaluates

the expression $counter + 1$ (6), and stores the result of the computation in the variable on the left, here *counter*. The effect of the statement is to increment *counter* by 1 from 5 to 6. We see later that incrementing by 1 is important when we need to count. In many situations, we need an old value of a variable to compute the new value of the same variable.

Quick Review Question 11 Write a segment to assign 34 to variable *time* and then to add 0.5 to *time*, changing its value.

Above, we do not need to display the value of *counter* after the assignment of 5 to the variable; we know the value of *counter* is 5. To suppress the output from a statement, we follow the statement with a **semicolon** (;), as follows:

```
counter = 5;
```

Quick Review Question 12 Repeat the previous question suppressing the output from the initial assignment.

User-defined Functions

Frequently, we wish to define our own functions that we can use again and again. As with variable names, by convention, we begin the name of such a **user-defined function** with a lowercase letter.

Suppose we wish to define the function $sqr(x) = x^2$ in *MATLAB*. In case *sqr* has a definition in memory from previous work, we clear any earlier definition of *sqr* with *clear* and the string '*sqr*', which has the name of the function in apostrophes, as follows:

```
clear('sqr')
```

To clear function definitions or variable values for several symbols, we separate strings with the symbols by commas, as follows:

```
clear('sqr', 'f')
```

A function definition can go into its own file for use later. However, a temporary function with a short expression definition is usually defined only in the workspace. Such a function is an **anonymous function**. For example, the definition of an anonymous function to return the square of numeric parameter is below. The associated **function handle**, *sqr*, provides a way of indirectly calling the anonymous function.

```
sqr = @(x) x*x;
```

After the function handle (here, *sqr*), an equal sign and the @ symbol, parentheses contain the **input parameter(s)** (here, *x*). Then, the definition in terms of the parameter, $x * x$, appears. (In the third tutorial, we see a better way to define this function so that

arguments can be arrays as well as numbers.) Then, from the command window we can call the function with an argument of 4, as follows:

```
sqr(4)
```

Quick Review Question 13 Define an anonymous function $f(x) = 3\sin(x - 1) + 2$. Evaluate the function at 5.

We can use symbols as well as numbers for a function's argument. For example, the following call to the function with handle *sqr* uses an undefined symbol, *z*, in the argument:

```
z = sym('z');
sqr(5 * z)
```

This call to the function returns the square of $5z$, which is $25z^2$.

Quick Review Question 14

- Clear any possible values for p and t .
- Define the population function $p(t) = 100e^{0.1t}$.
- Evaluate p at $t = 12$.

Online Documentation

We can obtain extensive online help under the **Help menu** by selecting **MATLAB Help**. We can click the **Search tab**, type the name of a term and click the **Go** button or can maneuver through the various menus to the desired help.

To find out information about a function, we can also type **help** followed by the name, such as the following statement to obtain information on the *MATLAB* function *log*:

```
help log
```

The system returns the following in response:

```
LOG    Natural logarithm.
LOG(X) is the natural logarithm of the elements of X.
Complex results are produced if X is not positive.

See also log1p, log2, log10, exp, logm.

Overloaded functions or methods (ones with the same name in other
directories)
    help sym/log.m

Reference page in Help browser
    doc log
```

Quick Review Question 15

- a. By typing a command, obtain help on the function *exp*.
- b. List two ways to obtain help on the function *exp* from the *Help* menu.

Displaying

Sometimes, particularly when doing error checking, we wish to display intermediate results. To do so, we can employ the *disp* function. To display the value of a variable without variable name and line feed, we can use the following form:

```
disp(variable)
```

Another form of the function call employs an array of strings, with the following format:

```
disp([string1, string2, ... ])
```

The command displays the strings (*string1*, *string2*, ...) with no spaces between them. For a **string constant**, which is the sequence of characters within apostrophes, *MATLAB* prints the characters between the apostrophes. For the value of a variable in such an array, we employ the function *num2str*, which converts the value of the variable to a string for display. The following segment displays string constant documentation along with the value of a variable, *timeOfExp*:

```
disp(['Time = ', num2str(timeOfExp), ' seconds'])
```

If *timeOfExp* has a value of 55.4, the output is as follows:

```
Time = 55.4 seconds
```

Quick Review Question 16 Write a statement to assign 3 to *t*. Then, employ *disp* and *num2str* to display "Velocity is", the result of the computation $-9.8 * t$, and "m/sec." For proper spacing, be sure to type blanks in the string constants after "is" and before "m/sec". Output for the *disp* command should be as follows:

```
Velocity is -29.4m/sec.
```

Looping

It is often advantageous to be able to execute a segment of code a number of times. For example, to obtain the velocity for each integer time ranging from 1 to 1000 seconds, it would be inconvenient for the user to have to execute one thousand times statements assigning a time and computing the corresponding velocity. Some method of automating the procedure is far more preferable. A segment of code that is executed repeatedly is called a **loop**.

Definition A **loop** is a segment of code that is executed repeatedly.

Several types of loops exist in *MATLAB*. When we know exactly how many times to execute the loop, **for** is often a good choice for implementing the loop. One form of the command is as follows:

```
for i = imin:imax
    expr
end
```

The loop's **index** or **loop variable** is *i*; and *i* takes on integer values from the loop's initial value, *imin*, through its terminal value, *imax*. For each value of *i*, the computer executes the body of the loop, which is *expr*, the statements between the line with *for* and **end**. For example, suppose, as the basis for a more involved segment, we wish to increment distance by 2.25 for 7 times. We initialize the distance variable, say *dist*, to be 0. Within a *for* loop that executes 7 times, we calculate the sum of *dist* and 2.25 and assign the result of the expression to *dist*, giving the variable an updated value. Because the *for* loop does not return a value and we are not printing intermediate values of *dist*, we display the final value of *dist* after the loop. The segment, which returns 15.75, is as follows:

```
dist = 0;

for i = 1:7
    dist = dist + 2.25;
end

dist
```

Quick Review Question 17 Write a segment to assign 1 to a variable *d* without displaying 1. In a loop that executes 10 times, change the value of *d* to be double what it was before the previous iteration. After the loop, type *d* so that *MATLAB* displays *d*'s final value. Before executing the loop, determine the final value so you can check your work.

In the next loop, we increment *dist* by 2.25, compute time as $\frac{24.5 - \sqrt{600.25 - 19.6dist}}{9.8}$, and then display distance and time. Because of its length, we break the *disp* command after a comma and use the **continuation symbol**, three consecutive periods (...). Without a continuation symbol, *MATLAB* attempts to execute an incomplete command. *MATLAB* expects a *for* loop to be on more than one line, however, and presents a fresh line without a prompt when we press return after "for *i* = 1:7".

```
dist = 0;

for i = 1:7
    dist = dist + 2.25;
    t = (24.5 - sqrt(600.25 - 19.6 * dist))/9.8;
    disp(['For distance = ', num2str(dist), ...
        ' time = ', num2str(t), ' seconds.']);
end
```

```
end
```

Subsequent output is as follows:

```
For distance = 2.25, time = 0.0935885 seconds.
For distance = 4.5, time = 0.190967 seconds.
For distance = 6.75, time = 0.292638 seconds.
For distance = 9., time = 0.399223 seconds.
For distance = 11.25, time = 0.511513 seconds.
For distance = 13.5, time = 0.630535 seconds.
For distance = 15.75, time = 0.75767 seconds.
```

Quick Review Question 18 This question is a variation of Quick Review Question 17. Write a segment to assign 1 to a variable d without displaying 1. In a loop that executes 10 times, change the value of d to be double what it was before the previous iteration and then print the value of d . Use a continuation symbol in the `disp` command. The output appears on 10 lines.

The previous example did not use the loop index in the loop's body. However, the following example of `disp` and `for` with an index i displays i and i -factorial ($i!$) with blanks between the values for i going from 1 through 9:

```
for i = 1:9
    disp([num2str(i), ' ', num2str(factorial(i))])
end
```

The following output shows each designated integer and its factorial:

```
1 1
2 2
3 6
4 24
5 120
6 720
7 5040
8 40320
9 362880
```

To start the display with the value of $0!$, which is 1, we indicate a beginning value of 0, as follows:

```
for i = 0:9
    disp([num2str(i), ' ', num2str(factorial(i))])
end
```

Quick Review Question 19 For this question, complete another version of the segment above that displays distance and time. In this version, do not initialize `dist`. Employ a loop with an index i that takes on integer values from 1 through 7. Within the loop, the value of `dist` is computed as $2.25i$. After replacing each `xxxxxxxxx` with the proper code, execute, and compare the results with the similar segment above.

```

for xxxxxxxxxxx
    dist = xxxxxxxxxxx
    t = (24.5 - sqrt(600.25 - 19.6 * dist))/9.8;
    disp(['For distance = ', num2str(dist), ...
         ' time = ', num2str(t), ' seconds.']);
xxxxxxxxxxxxx

```

Quick Review Question 20

- Clear possible values for the symbols $qrq20$ and x .
- Define the function $qrq20(x) = \ln(3x + 2)$. Recall that \log is the *MATLAB* function for the natural logarithm.
- Write a loop that prints the value of k and $qrq20(k)$ for k taking on integer values from 1 through 8.

Plotting

We employ the *plot* command for graphing two-dimensional functions. First, we establish a sequence of values for the independent variable, such as t . We employed **sequences** in the *for* loop by giving an index's integer minimum and maximum values with a **colon separator**. For example, 0:9 indicates the sequence 0, 1, ..., 9. For a smooth display of a graph, we need to plot the graph at additional points. To indicate a step size of 0.1, we place the step size between the minimum and maximum values, which are not necessarily integers, with colon separators, as in -1:0.1:2. This indicates a sequence of numbers from -1 to 2 that increases in steps of size 0.1, namely -1, -0.9, -0.8, ..., 0, 0.1, 0.2, ..., 1.8, 1.9, 2.0. The following statement assigns such a sequence to t :

```
t = -1 : 0.1 : 2;
```

The basic form of the command gives the independent variable (such as x) and the function (such as g) to graph, as follows:

```
plot(x, g(x))
```

For example, the following command graphs $\cos(t)$ with t varying from -1 to 2 by 0.1:

```
t = -1:0.1:2;
plot(t, cos(t))
```

MATLAB displays the resulting graphics in a *figure* window.

Quick Review Question 21

Graph $e^{\sin(x)}$ from -3 to 3.

We can indicate additional options, and *MATLAB* revises the display to reflect our changes. For example, *xlabel* and *ylabel* options generate axes labels, which should appear in all scientific graphics. The command that follows plots the graph and labels the horizontal and vertical axes with t and y , respectively:

```
xlabel('t')
```

```
ylabel('y')
```

Quick Review Question 22 Adjust the answer to the previous Quick Review Question to label the x and y axes.

To plot several functions on the same graph using *plot*, for the arguments we have a sequence of alternating independent variables and functions, as follows:

```
plot(t, cos(t), t, 3*t + 4);
```

Quick Review Question 23 Adjust the answer to the previous Quick Review Question to plot $e^{\sin(x)}$ and $\sin(x)$ on the same graph.

In future *MATLAB* tutorials, we consider other plotting options, but the basic *plot* command with the *xlabel* and *ylabel* options supports much modeling work.

Differentiation

See Module 2.3 on "Rate of Change" for a discussion of differentiation.

MATLAB's Symbolic Math Toolboxes provide the function *diff* for taking the derivative. For example, consider the following function for the height of a ball thrown into the air:

$$y = -4.9t^2 + 15t + 11$$

With symbol t , we define y , as follows in *MATLAB*:

```
syms t
y = -4.9 * t * t + 15 * t + 11;
```

The following call to *diff* returns the **derivative**, $-49/5*t+15$ or $15 - 9.8t$, where t is a symbol:

```
diff(y)
```

To evaluate y at 1, we can employ the *subs* command to **substitute** 1 for t , as follows:

```
subs(y, 1)
```

For the derivative at 1, which is $y'(1) = 5.20$, we save the derivative in a variable and call *subs*, as follows:

```
dy_dt = diff(y);
subs(dy_dt, 1)
```

Without assigning an expression to a variable, such as y , we can employ the *diff* command to compute the derivative. Thus, to take the derivative of $-4.9t^2 + 15t + 11$ with respect to t , we can employ the following command:

```
diff( -4.9 * t * t + 15 * t + 11 )
```

Quick Review Question 24 Give *MATLAB* code to do the following:

- Make x a symbol.
- Define $y = 2.9 \sin(0.03x)$.
- Evaluate the derivative of y with respect to x .
- Evaluate the derivative of y at 35.

Solving Differential Equations

See Module 2.3 on "Rate of Change" for a discussion of differential equations.

We can employ the *MATLAB* function *dsolve* from the Symbolic Math Toolboxes to solve a differential equation or a system of differential equations. For example, suppose we wish to solve the differential equation $F'(t) = -t^2 + 10t + 24$ with initial condition $F(0) = 30$. With apostrophes surrounding each equation and *DF* expressing the derivative of F , $F'(t)$, the following command solves the differential equation with initial condition:

```
dsolve('DF = -t^2 + 10*t + 24', 'F(0) = 30')
```

MATLAB returns the following solution:

```
-1/3*t^3+5*t^2+24*t+30
```

Quick Review Question 25 Write a *MATLAB* command to solve the differential equation $P'(t) = 0.1P(t)$ with initial condition $P(0) = 100$. The module on "Unconstrained Growth" uses this differential equation.

Integration

See Module 2.4 on "Fundamental Concepts of Integral Calculus" for a discussion of integration.

We can request that *MATLAB* with the Symbolic math Toolboxes evaluate definite integrals and indefinite integrals using the *int* command. The indefinite integral $\int (-t^2 + 10t + 24) dt$ is as follows:

```
int(-t^2 + 10*t + 24)
```

The definite integral, such as $\int_0^5 (-t^2 + 10t + 24) dt$, has additional arguments for the limits of integration, such as follows:

```
int(-t^2 + 10*t + 24, 1, 5)
```

Quick Review Question 26

- a. Designate that x is a symbol.
- b. Obtain the indefinite integral of $\sin^2(x)$.
- c. Obtain the definite integral of $\sin^2(x)$ from 0 to 2π .