1

Introduction to MATLAB® and Its Graphics Capabilities

1.1 Getting Started

MATLAB can be thought of as a library of programs that will prove very useful in solving many electrical engineering computational problems. MATLAB is an ideal tool for numerically assisting you in obtaining answers, which is a major goal of engineering analysis and design. This program is very useful in circuit analysis, device design, signal processing, filter design, control system analysis, antenna design, microwave engineering, photonics engineering, computer engineering, and all other sub-fields of electrical engineering. It is also a powerful graphic and visualization tool.

The first step in using MATLAB is to know how to call it. It is important to remember that although the front-end and the interfacing for machines with different operating systems are sometimes different, once you are inside MATLAB, all programs and routines are written in the same manner. Only those few commands that are for file management and for interfacing with external devices such as printers may be different for different operating systems.

After entering MATLAB, you should see the prompt **`>>`**, which means the program interpreter is waiting for you to enter instructions. (Remember to press the Return key at the end of each line that you enter.)

Now type **`clf`**. This command creates a graph window (if one does not already exist) or clears an existing graph window.

Because it is impossible to explain the function of every MATLAB command within this text, how would you get information on a certain command syntax? The MATLAB program has extensive help documentation available with simple commands. For example, if you wanted help on a function called **`roots`** (we will use this function often), you would type **`help roots`**.

Note that the help facility cross-references other functions that may have related uses. This requires that you know the function name. If you want an idea of the available help files in MATLAB, type **`help`**. This gives you a list of topics included in MATLAB. To get help on a particular topic such as the Optimization Toolbox, type **`help toolbox/optim`**. This gives you a list of

all relevant functions pertaining to that area. Now you may type **help** for any function listed. For example, try **help fmin**.

1.2 Basic Algebraic Operations and Functions

The MATLAB environment can be used, on the most elementary level, as a tool to perform simple algebraic manipulations and function evaluations.

Example 1.1

Exploring the calculator functions of MATLAB. The purpose of this example is to show how to manually enter data and how to use basic MATLAB algebraic operations. Note that the statements will be executed immediately after they are typed and entered (no equal sign is required).

Type and enter the text that follows the **>>** prompt to find out the MATLAB responses to the following:

```
2+2
5^2
2*sin(pi/4)
```

The last command gave the sine of $\pi/4$. Note that the argument of the function was enclosed in parentheses directly following the name of the function. Therefore, if you wanted to find $\sin^3(\pi/4)$, the proper MATLAB syntax would be

```
sin(pi/4)^3
```

To facilitate its widespread use, MATLAB has all the standard elementary mathematical functions as built-in functions. Type **help elfun**, which is indexed in the main help menu to get a listing of some of these functions. Remember that this is just a small sampling of the available functions.

```
help elfun
```

The response to the last command will give you a large list of these elementary functions, some of which may be new to you, but all of which will be used in your future engineering studies, and explored in later chapters of this book.

Example 1.2

Assigning and calling values of parameters. In addition to inputting data directly to the screen, you can assign a symbolic constant or constants to rep-

resent data and perform manipulations on them. For example, enter and note the answer to each of the following:

```
a=2
b=3
c=a+b
d=a*b
e=a/b
f=a^3/b^2
g=a+3*b^2
```

Question: From the above, can you deduce the order in which MATLAB performs the basic operations?

In-Class Exercise

Pb. 1.1 Using the above values of a and b, find the values of:

a. $h = \sin(a)\,\sin(b)$

b. $i = a^{1/3}b^{3/7}$

c. $j = \sin^{-1}(a/b) = \arcsin(a/b)$

1.3 Plotting Points

In this chapter section, you will learn how to use some simple MATLAB graphics commands to plot points. We use these graphics commands later in the text for plotting functions and for visualizing their properties. To view all the functions connected with 2-dimensional graphics, type:

```
help plot
```

All graphics functions connected with 3-dimensional graphics can be looked up by typing

```
help plot3
```

A point P in the x-y plane is specified by two coordinates. The x-coordinate measures the horizontal distance of the point from the y-axis, while the y-coordinate measures the vertical distance above the x-axis. These coordi-

nates are called Cartesian coordinates, and any point in the plane can be described in this manner. We write for the point, P(x, y).

Other representations can also be used to locate a point with respect to a particular set of axes. For example, in the polar representation, the point is specified by an r-coordinate that measures the distance of the point from the origin, while the θ-coordinate measures the angle which the line passing through the origin and this point makes with the x-axis.

The purpose of the following two examples is to learn how to represent points in a plane and to plot them using MATLAB.

Example 1.3

Plot the point P(3, 4).

Solution: Enter the following:

```
x1=3;
y1=4;
plot(x1,y1,'*')
```

Note that the semicolon is used in the above commands to suppress the echoing of the values of the inputs. The **`'*'`** is used to mark the point that we are plotting. Other authorized symbols for point displays include **`'o'`**, **`'+'`**, **`'x'`**, … the use of which is detailed in **`help plot`**.

Example 1.4

Plot the second point, R(2.5, 4) on the graph while keeping point P of the previous example on the graph.

Solution: If we went ahead, defined the coordinates of R, and attempted to plot the point R through the following commands:

```
x2=2.5;
y2=4;
plot(x2,y2,'o')
```

we would find that the last plot command erases the previous plot output.

Thus, what should we do if we want both points plotted on the same graph? The answer is to use the **`hold on`** command after the first plot.

The following illustrates the steps that you should have taken instead of the above:

```
hold on
x2=2.5;
```

```
y2=4;
plot(x2,y2,'o')
hold off
```

The **hold off** turns off the **hold on** feature.

NOTES

1. There is no limit to the number of plot commands you can type before the hold is turned off.
2. An alternative method for viewing multiple points on the same graph is available: we may instead, following the entering of the values of **x1, y1, x2, y2**, enter:

```
plot(x1,y1,'*',x2,y2,'o')
```

This has the advantage, in MATLAB, of assigning automatically a different color to each point.

1.3.1 Axes Commands

You may have noticed that MATLAB automatically adjusts the scale on a graph to accommodate the coordinates of the points being plotted. The axis scaling can be manually enforced by using the command **axis([xmin xmax ymin ymax])**. Make sure that the minimum axis value is less than the maximum axis value or an error will result.

In addition to being able to adjust the scale of a graph, you can also change the aspect ratio of the graphics window. This is useful when you wish to see the correct x to y scaling. For example, without this command, a circle will look more like an ellipse.

Example 1.5

Plot the vertices of a square, keeping the geometric proportions unaltered.

Solution: Enter the following:

```
x1=-1;y1=-1;x2=1;y2=-1;x3=-1;y3=1;x4=1;y4=1;
plot(x1,y1,'o',x2,y2,'o',x3,y3,'o',x4,y4,'o')
axis([-2 2 -2 2])
axis square                %square shape
```

Note that prior to the **axis square** command, the square looked like a rectangle. If you want to go back to the default aspect ratio, type **axis normal**. The **%** symbol is used so that you can type comments in your program. Comments following the **%** symbol are ignored by the MATLAB interpreter.

1.3.2 Labeling a Graph

To add labels to your graph, the functions **xlabel**, **ylabel**, and **title** can be used as follows:

```
xlabel('x-axis')
ylabel('y-axis')
title('points in a plane')
```

If you desire to add a caption anywhere in the graph, you can use the MATLAB command **gtext('caption')** and place it at the location of your choice, on the graph, by clicking the mouse when the crosshair is properly centered there.

1.3.3 Plotting a Point in 3-D

In addition to being able to plot points on a plane (2-D space), MATLAB is also able to plot points in a three-dimensional space (3-D space). For this, we utilize the **plot3** function.

Example 1.6
Plot the point P(3, 4, 5).

Solution: Enter the following commands:

```
x1=3; y1=4; z1=5;
plot3(x1,y1,z1,'*')
```

You can also plot multiple points in a 3-D space in exactly the same way as you did on a plane. Axis adjustment can still be used, but the vector input into the **axis** command must now have six entries, as follows:

```
axis([xmin xmax ymin ymax zmin zmax])
```

You can similarly label your 3-D figure using **xlabel**, **ylabel**, **zlabel**, and **title**.

1.4 M-files

In the last section, we found that to complete a figure with a caption, we had to enter several commands one by one in the command window. Typing

errors will be time-consuming to fix because if you are working in the command window, you need to retype all or part of the program. Even if you do not make any mistakes (!), all of your work may be lost if you inadvertently quit MATLAB and have not taken the necessary steps to save the contents of the important program that you just finished developing. To preserve large sets of commands, you can store them in a special type of file called an *M-file*.

MATLAB supports two types of *M-file*s: *script* and *function M-file*s. To hold a large collection of commands, we use a *script M-file*. The *function M-file* is discussed in Chapter 3. To make a *script M-file*, you need to open a file using the built-in MATLAB editor. For both Macs and PCs, first select New from the file menu. Then select the *M-file* entry from the pull-down menu. After typing the *M-file* contents, you need to save the file:

For Macs and PCs, select the **save as** command from the file window. A field will pop up in which you can type in the name you have chosen for this file (make sure that you do not name a file by a mathematical abbreviation, the name of a mathematical function, or a number). Also make sure that the file name has a **.m** extension added at the end of its name.

For Macs, save the file in a user's designated volume.

For PCs, save the file in the default (bin) subdirectory.

To run your *script M-file*, just type the filename (omitting the **.m** extension at its end) at the MATLAB prompt.

Example 1.7

For practice, go to your file edit window to create the following file that you name **myfile.m**.

```
clear, clf
x1=1;y1=.5;x2=2;y2=1.5;x3=3;y3=2;
plot(x1,y1,'o',x2,y2,'+',x3,y3,'*')
axis([0 4 0 4])
xlabel('xaxis')
ylabel('yaxis')
title('3points in a plane')
```

After creating and saving **myfile.m**, go to the MATLAB command window and enter **myfile**. MATLAB will execute the instructions in the order of the statements stored in your **myfile.m** file.

1.5 MATLAB Simple Programming

1.5.1 Iterative Loops

The power of computers lies in their ability to perform a large number of repetitive calculations. To do this without entering the value of a parameter or variable each time that these are changed, all computer languages have control structures that allow commands to be performed and controlled by counter variables, and MATLAB is no different. For example, the MATLAB "**for**" loop allows a statement or a group of statements to be repeated.

Example 1.8
Generate the square of the first ten integers.

Solution: Edit and execute the the following *script M-file:*

```
for m=1:10
x(m)=m^2;
end;
```

In this case, the number of repetitions is controlled by the index variable **m**, which takes on the values **m** = 1 through **m** = 10 in intervals of 1. Therefore, ten assignments were made. What the above loop is doing is sequentially assigning the different values of **m^2** (i.e., m^2) in each element of the "**x**-array." An array is just a data structure that can hold multiple entries. An array can be 1-D such as in a vector, or 2-D such as in a matrix. More will be said about vectors and matrices in subsequent chapters. At this time, think of the 1-D and 2-D arrays as pigeonholes with numbers or ordered pair of numbers respectively assigned to them.

To find the value of a particular slot of the array, such as slot 3, enter:

```
x(3)
```

To read all the values stored in the array, type:

```
x
```

Question: What do you get if you enter **m**?

1.5.2 If-Else-End Structures

If a sequence of commands must be conditionally evaluated based on a relational test, the programming of this logical relationship is executed with some variation of an **if-else-end** structure.

A. The simplest form of this structure is:

if *expression*

commands evaluated if expression is True

else

commands evaluated if expression is False

end

NOTES

1. The commands between the **if** and **else** statements are evaluated if all elements in the expression are true.
2. The conditional expression uses the Boolean logical symbols **&** (and), **|** (or), and **~** (not) to connect different propositions.

Example 1.9

Find for integer $0 < a \leq 10$, the values of *C*, defined as follows:

$$C = \begin{cases} ab & \text{for } a > 5 \\ \frac{3}{2} & ab \text{ for } a \leq 5 \end{cases}$$

and $b = 15$.

Solution: Edit and execute the following *script M-file:*

```
for a=1:10
b=15;
  if a>5
    C(a)=a*b;
  else
    C(a)=(a*b)*(3/2);
  end
end
```

Check that the values of C that you obtain by typing **C** are:

```
22.5  45  67.5  90  112.50  90  105  120  135  150
```

B. When there are three or more alternatives, the **if-else-end** structure takes the form:

if *expression 1*

Commands 1 evaluated if expression 1 is True

```
elseif expression 2
   Commands 2 evaluated if expression 2 is True
elseif  expression 3
   Commands 3 evaluated if expression 3 is True
…
else
   Commands evaluated if no other expression is True
end
```

In this form, only the commands associated with the first True expression encountered are evaluated; ensuing relational expressions are not tested.

1.5.2.1 Alternative Syntax to the `if` *Statement*

As an alternative to the **if** syntax, we can use, in certain instances, Boolean expressions to specify an expression in different domains. For example, **`(x>=l)`** has the value 1 if **`x`** is larger than or equal to 1 and zero otherwise; and **`(x<=h)`** is equal to 1 when **`x`** is smaller than or equal to **`h`**, and zero otherwise.

The relational operations allowed inside the parentheses are: **`==`**, **`<=`**, **`>=`**, **`~=`**, **`<`**, **`>`**.

Homework Problem

Pb. 1.2 For the values of integer a going from 1 to 10, using separately the methods of the **`if`** syntax and the Boolean alternative expressions, find the values of C if:

$$\begin{aligned} C &= a^2 && \text{for } a < 3 \\ C &= a + 5 && \text{for } 3 \le a < 7 \\ C &= a && \text{for } a \ge 7 \end{aligned}$$

Use the **`stem`** command to graphically show C.

1.6 Array Operations

In the above examples, we used **`for`** loops repeatedly. However, this kind of loop-programming is very inefficient and must be avoided as much as possi-

ble in MATLAB. In fact, ideally, a good MATLAB program will always minimize the use of loops because MATLAB is an interpreted language — not a compiled one. As a result, any looping process is very inefficient. Nevertheless, at times we use the **for** loops, when necessitated by pedagogical reasons.

To understand array operations more clearly, consider the following:

```
a=1:3 % a starts at 1, goes to 3 in increments of 1.
```

If the increment is not 1, you must specify the increment; for example:

```
b=2:2:6 % b starts at 2, goes to 6 in increments of 2
```

To distinguish arrays operations from either operations on scalars or on matrices, the symbol for multiplication becomes **.***, that of division **./**, and that of exponentiation **.^**. Thus, for example:

```
c=a.*b % takes every element of a and multiplies
  % it by the element of b in the same array location
```

Similarly, for exponentiation and division:

```
d=a.^b
e=a./b
```

If you try to use the regular scalar operations symbols, you will get an error message.

Note that array operations such as the above require that the two arrays have the same length (i.e., the same number of elements). To verify that two arrays have the same number of elements (dimension), use the **length** command. Thus, to find the length of **a** and **b**, enter:

```
length(a)
length(b)
```

NOTE The expression **x=linspace(0,10,200)** is also the generator for an x-array with first element equal to 0, a last element equal to 10, and having 200 equally spaced points between 0 and 100. Here, the number of points rather than the increment is specified; that is, **length(x)=200**.

1.7 Curve and Surface Plotting

Review the sections of the Supplement pertaining to lines, quadratic functions, and trigonometric functions before proceeding further.

1.7.1 *x-y* Parametric Plot

Now edit another *M-file* called **myline.m** as follows and execute it.

```
N=10;
for m=1:N
  x(m)=m;
  y(m)=2*m+3;
end
plot(x,y)
```

After executing the *M-file* using **myline**, you should see a straight line connecting the points (1, 5) and (10, 23). This demonstration shows the basic construct for creating two arrays and plotting the points with their x-coordinate from a particular location in one array and their y-coordinate from the same location in the second array. We say that the **plot** command here plotted the y-array vs. the x-array.

We note that the points are connected by a continuous line making a smooth curve; we say that the program graphically interpolated the discrete points into a continuous curve. If we desire to see additionally the individual points corresponding to the values of the arrays, the last command should be changed to:

```
plot(x,y,x,y,'o')
```

Example 1.10

Plot the two curves $y_1 = 2x + 3$ and $y_2 = 4x + 3$ on the same graph.

Solution: Edit and execute the following *script M-file:*

```
for m=1:10
  x(m)=m;
  y1(m)=2*m+3;
  y2(m)=4*m+3;
end
plot(x,y1,x,y2)
```

or better

```
m=1:10;
x=m;
y1=2*m+3;
y2=4*m+3;
plot(x,y1,x,y2)
```

Finally, note that you can separate graphs in one figure window. This is done using the **subplot** function in MATLAB. The arguments of the subplot function are **subplot(m,n,p)**, where m is the number of rows partitioning the graph, n is the number of columns, and p is the particular subgraph chosen (enumerated through the left to right, top to bottom convention).

1.7.1.1 Demonstration: Plotting Multiple Figures within a Figure Window

Using the data obtained in the previous example, observe the difference in the partition of the page in the following two sets of commands:

```
subplot(2,1,1)
plot(x,y1)
subplot(2,1,2)
plot(x,y2)
```

and

```
clf
subplot(1,2,1)
plot(x,y1)
subplot(1,2,2)
plot(x,y2)
```

1.7.2 More on Parametric Plots in 2-D

In the preceding subsection, we generated the x- and y-arrays by first writing the x-variable as a linear function of a parameter, and then expressed the dependent variable y as a function of that same parameter. What we did is that, instead of thinking of a function as a relation between an independent variable x and a dependent variable y, we thought of both x and y as being dependent functions of a third independent parameter. This method of curve representation, known as the parametric representation, is described by $(x(t), y(t))$, where the parameter t varies over some finite domain $(t_{\min}, t_{\max})$. Note, however, that in the general case, unlike the examples in the previous chapter subsection, the independent variable x need not be linear in the parameter, nor is the process of parametrization unique.

Example 1.11

Plot the trigonometric circle.

Solution: Recalling that the x-coordinate of any point on the trigonometric circle has the cosine as x-component and the sine as y-component, the generation of the trigonometric circle is immediate:

```
th=linspace(0,2*pi,101)
x=cos(th);
y=sin(th);
```

```
plot(x,y)
axis square
```

The parametric representation of many common curves is our next topic of interest. The parametric representation is defined such that if x and y are continuous functions of t over the interval I, we can describe a curve in the x-y plane by specifying:

$$C: x = x(t),\ y = y(t), \text{ and } t \in I$$

More Examples:

In the following examples, we want to identify the curves $f(x, y) = 0$ corresponding to each of the given parametrizations.

Example 1.12

C: $x = 2t - 1$, $y = t + 1$, and $0 < t < 2$. The initial point is at $x = -1$, $y = 1$, and the final point is at $x = 3$, $y = 3$.

Solution: The curve $f(x, y) = 0$ form can be obtained by noting that:

$$2t - 1 = x \Rightarrow t = (x + 1)/2$$

Substitution into the expression for y results in:

$$y = \frac{x}{2} + \frac{3}{2}$$

This describes a line with slope $1/2$ crossing the x-axis at x = –3.

Question: Where does this line cross the y-axis?

Example 1.13

C: $x = 3 + 3\cos(t)$, $y = 2 + 2\sin(t)$, and $0 < t < 2\pi$. The initial point is at $x = 6$, $y = 2$, and the final point is at $x = 6$, $y = 2$.

Solution: The curve $f(x, y) = 0$ can be obtained by noting that:

$$\sin(t) = \frac{y-2}{2} \quad \text{and} \quad \cos(t) = \frac{x-3}{3}$$

Using the trigonometric identity $\cos^2(t) + \sin^2(t) = 1$, we deduce the following equation:

$$\frac{(y-2)^2}{2^2}+\frac{(x-3)^2}{3^2}=1$$

This is the equation of an ellipse centered at $x = 3$, $y = 2$ and having major and minor radii equal to 3 and 2, respectively.

Question 1: What are the coordinates of the foci of this ellipse?

Question 2: Compare the above curve with the curve defined through:

$$x = 3 + 3\cos(2t),\ y = 2 + 2\sin(2t),\ \text{and}\ 0 < t < 2\pi$$

What conclusions can you draw from your answer?

In-Class Exercises

Pb. 1.3 Show that the following parametric equations:

$$x = h + a\sec(t),\ y = k + b\tan(t),\ \text{and}\ -\pi/2 < t < \pi/2$$

are those of the hyperbola also represented by the equation:

$$\frac{(x-h)^2}{a^2}-\frac{(y-k)^2}{b^2}=1$$

Pb. 1.4 Plot the hyperbola represented by the parametric equations of **Pb. 1.3**, with $h = 2$, $k = 2$, $a = 1$, $b = 2$. Find the coordinates of the vertices and the foci. (*Hint:* One branch of the hyperbola is traced for $-\pi/2 < t < \pi/2$, while the other branch is traced when $\pi/2 < t < 3\pi/2$.)

Pb. 1.5 The parametric equations of the cycloid are given by:

$$x = R\omega t + R\sin(\omega t),\ y = R + R\cos(\omega t),\ \text{and}\ 0 < t$$

Show how this parametric equation can be obtained by following the kinematics of a point attached to the outer rim of a wheel that is uniformly rolling, without slippage, on a flat surface. Relate the above parameters to the linear speed and the radius of the wheel.

Pb. 1.6 Sketch the curve C defined through the following parametric equations:

$$x(t)=\begin{cases} t+2 & \text{for } -3\le t\le -1 \\ +1-\dfrac{1}{\sqrt{3}}\tan\left(\dfrac{\pi}{3}(1-t^2)\right) & \text{for } -1<t<0 \\ -1+\dfrac{1}{\sqrt{3}}\tan\left(\dfrac{\pi}{3}(1-t^2)\right) & \text{for } 0<t<1 \end{cases}$$

$$y(t)=\begin{cases} 0 & \text{for } -3\le t\le -1 \\ \dfrac{1}{\sqrt{3}}\tan\left(\dfrac{\pi}{3}(1-t^2)\right) & \text{for } -1<t<0 \\ \dfrac{1}{\sqrt{3}}\tan\left(\dfrac{\pi}{3}(1-t^2)\right) & \text{for } 0<t<1 \end{cases}$$

Homework Problems

The following set of problems provides the mathematical basis for understanding the graphical display on the screen of an oscilloscope, when in the *x*-*y* mode.

Pb. 1.7 To put the quadratic expression

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

in standard form (i.e., to eliminate the *x*-*y* mixed term), make the transformation

$$x = x'\cos(\theta) - y'\sin(\theta)$$

$$y = x'\sin(\theta) + y'\cos(\theta)$$

Show that the mixed term is eliminated if $\cot(2\theta) \equiv \dfrac{(A-C)}{B}$.

Pb. 1.8 Consider the parametric equations

$$C\text{: } x = a\cos(t),\ y = b\sin(t+\varphi), \text{ and } 0 < t < 2\pi$$

where the initial point is at $x = a$, $y = b\sin(\varphi)$, and the final point is at $x = a$, $y = b\sin(\varphi)$.

a. Obtain the equation of the curve in the form $f(x, y) = 0$.

b. Using the results of **Pb. 1.7**, prove that the ellipse inclination angle is given by:

$$\cot(2\theta) \equiv \frac{(a^2-b^2)}{2ab\sin(\varphi)}$$

Pb. 1.9 If the parametric equations of a curve are given by:

$$C: x = \cos(t),\ y = \sin(2t), \text{ and } 0 < t < 2\pi$$

where the initial point is at $x = 1$, $y = 0$, and the final point is at $x = 1$, $y = 0$.

The curve so obtained is called a Lissajous figure. It has the shape of a figure 8 with two nodes in the x-direction and only one node in the y-direction.

What do you think the parametric equations should be if we wanted m nodes on the x-axis and n nodes on the y-axis? Test your hypothesis by plotting the results.

1.7.3 Plotting a 3-D Curve

Our next area of exploration is plotting 3-D curves.

Example 1.14
Plot the helix.

Solution: To plot a helical curve, we can imagine initially that a point is revolving at a uniform speed around the perimeter of a circle. Now imagine that as the circular motion is continuing, the point is moving away from the x-y plane at some constant linear speed. The parametric representation of this motion can be implemented in MATLAB through the following:

```
for m=1:201
  th(m)=2*pi*.01*(m-1);
  x(m)=cos(th(m));
  y(m)=sin(th(m));
  z(m)=th(m);
end
plot3(x,y,z)
```

In-Class Exercises

Pb. 1.10 In the helix of Example 1.14, what is the vertical distance (the pitch) between two consecutive helical turns. How can you control this distance? Find two methods of implementation.

Pb. 1.11 If instead of a circle in 2-D, as in the helix, the particle describes in 2-D a Lissajous pattern having two nodes in the y-direction and three nodes

in the x-direction, assuming that the z-parametric equation remains the same, show the resulting 3-D trajectory.

Pb. 1.12 What if $z(t)$ is periodic in t? For example, $z(t) = \cos(t)$ or $z(t) = \cos(2t)$, while the 2-D motion is still circular. Show the 3-D trajectory.

In Example 1.14, we used the **for** loop to generate the dependent arrays for the helix; but as pointed out previously, a more efficient method to program the helix is in the array notation, as follows:

```
th=[0:.01:2]*2*pi;
x=cos(th);
y=sin(th);
z=th;
plot3(x,y,z)
```

1.7.4 Plotting a 3-D Surface

We now explore the two different techniques for rendering, in MATLAB, 3-D surface graphics: the mesh and the contour representations.

- A function of two variables $z = f(x, y)$ represents a surface in 3-D geometry; for example:

$$z = ax + by + c$$

  represents a plane that crosses the vertical axis (z-axis) at c.
- There are essentially two main techniques in MATLAB for viewing surfaces: the **mesh** function and the **contour** function.
- In both techniques, we must first create a 2-D array structure (like a checkerboard) with the appropriate x- and y-values. To implement this, we use the MATLAB **meshgrid** function.
- The z-component is then expressed in the variables assigned to implement the **meshgrid** command.
- We then plot the function with either the **mesh** command or the **contour** command. The **mesh** command gives a 3-D rendering of the surface, while the **contour** command gives contour lines, wherein each contour represents the locus of points on the surface having the same height above the x-y plane. This last rendering technique is that used by mapmakers to represent the topography of a terrain.

1.7.4.1 Surface Rendering

Example 1.15

Plot the sinc function whose equation is given by:

$$z = \frac{\sin\left(\sqrt{x^2 + y^2}\right)}{\sqrt{x^2 + y^2}}$$

over the domain $-8 < x < 8$ and $-8 < y < 8$.

Solution: The implementation of the mesh rendering follows:

```
x=[-8:.1:8];
y=[-8:.1:8];
[X,Y]=meshgrid(x,y);
R=sqrt(X.^2+Y.^2)+eps;
Z=sin(R)./R;
mesh(X,Y,Z)
```

The variable **eps** is a tolerance number = 2^{-52} used for determining expressions near apparent singularities, to avoid numerical division by zero.

To generate a contour plot, we replace the last command in the above by:

```
contour(X,Y,Z,50) % The fourth argument specifies
  % the number of contour lines to be shown
```

If we are interested only in a particular contour level, for example, the one with elevation Z_0, we use the contour function with an option, as follows:

```
contour(X,Y,Z,[Z₀ Z₀])
```

Occasionally, we might be interested in displaying simultaneously the mesh and contour rendering of a surface. This is possible through the use of the command **meshc.** It is the same as the **mesh** command except that a contour plot is drawn beneath the mesh.

Preparatory Activity: Look in your calculus book for some surfaces equations, such as those of the hyperbolic paraboloid and the elliptic paraboloid and others of your choice for the purpose of completing **Pb. 1.16** of the next in-class activity.

In-Class Exercises

Pb. 1.13 Use the **`contour`** function to graphically find the locus of points on the above sinc surface that are 1/2 units above the *x*-*y* plane (i.e., the surface intersection with the $z = 1/2$ plane).

Pb. 1.14 Find the *x*-*y* plane intersection with the following two surfaces:

$$z_1 = 3 + x + y$$

$$z_2 = 4 - 2x - 4y$$

Pb. 1.15 Verify your answers to **Pb. 1.14** with that which you would obtain analytically for the shape of the intersection curves of the surfaces with the *x*-*y* plane. Also, compute the coordinates of the point of intersection of the two obtained curves. Verify your results graphically.

Pb. 1.16 Plot the surfaces that you have selected in your preparatory activity. Look in the help folder for the **`view`** command to learn how to view these surfaces from different angles.

1.8 Polar Plots

MATLAB can also display polar plots. In the first example, we draw an ellipse of the form $r = 1 + \varepsilon \cos(\theta)$ in a polar plot; other shapes are given in the other examples.

Example 1.16

Plot the ellipse in a polar plot.

Solution: The following sequence of commands plot the polar plot of an ellipse with $\varepsilon = 0.2$:

```
th=0:2*pi/100:2*pi;
rho=1+.2*cos(th);
polar(th,rho)
```

The shape you obtain may be unfamiliar; but to verify that this is indeed an ellipse, view the curve in a Cartesian graph. For that, you can use the MATLAB polar to Cartesian converter **`pol2cart`**, as follows:

```
[x,y]=pol2cart(th,rho);
plot(x,y)
axis equal
```

Example 1.17

Graph the polar plot of a spiral.

Solution: The equation of the spiral is given by:

$$r = a\theta$$

Its polar plot can be viewed by executing the following *script M-file* ($a = 3$):

```
th=0:2*pi/100:2*pi;
rho=3*th;
polar(th,rho)
```

In-Class Exercises

Pb. 1.17 Prove that the polar equation $r = 1 + \varepsilon \cos(\theta)$, where ε is always between –1 and 1, results in an ellipse. (*Hint:* Relate ε to the ratio between the semi-major and semi-minor axis.) It is worth noting that the planetary orbits are usually described in this manner in most astronomy books.

Pb. 1.18 Plot the three curves described by the following polar equations:

$$r = 2 - 2\sin(\theta),\ \ r = 1 - \sqrt{2}\sin(\theta),\ \ r = \sqrt{2\sin(2\theta)}$$

Pb. 1.19 Plot:

$$r = \sin(2\theta)\cos(2\theta)$$

The above gives a flower-type curve with eight petals. How would you make a flower with 16 petals?

Pb. 1.20 Plot:

$$r = \sin^2(\theta)$$

This two-lobed structure shows the power distribution of a simple dipole antenna. Note the directed nature of the radiation. Can you increase the directivity further?

Pb. 1.21 Acquaint yourself with the polar plots of the following curves: (choose first $a = 1$, then experiment with other values).

a. Straight lines: $r = \dfrac{1}{\cos(\theta) + a\sin(\theta)}$ for $0 \le \theta \le \dfrac{\pi}{2}$

b. Cissoid of Diocles: $r = a\dfrac{\sin^2(\theta)}{\cos(\theta)}$ for $-\dfrac{\pi}{3} \le \theta \le \dfrac{\pi}{3}$

c. Strophoid: $r = \dfrac{a\cos(2\theta)}{\cos(\theta)}$ for $-\dfrac{\pi}{3} \le \theta \le \dfrac{\pi}{3}$

d. Folium of Descartes: $r = \dfrac{3a\sin(\theta)\cos(\theta)}{\sin^3(\theta) + \cos^3(\theta)}$ for $-\dfrac{\pi}{6} \le \theta \le \dfrac{\pi}{2}$

1.9 Animation

A very powerful feature of MATLAB is its ability to render an animation. For example, suppose that we want to visualize the oscillations of an ordinary spring. What are the necessary steps to implement this objective?

1. Determine the parametric equations that describe the curve at a fixed time. In this instance, it is the helix parametric equations as given earlier in Example 1.14.
2. Introduce the time dependence in the appropriate curve parameters. In this instance, make the helix pitch to be oscillatory in time.
3. Generate 3-D plots of the curve at different times. Make sure that your axis definition includes all cases.
4. Use the **movie** commands to display consecutively the different frames obtained in step 3.

The following *script M-file* implements the above workplan:

```
th=0:pi/60:32*pi;
a=1;
A=0.25;
w=2*pi/15;
M=moviein(16);
  for t=1:16;
  x=a*cos(th);
```

```
    y=a*sin(th);
    z=(1+A*cos(w*(t-1)))*th;
    plot3(x,y,z,'r');
    axis([-2 2 -2 2 0 40*pi]);
    M(:,t)=getframe;
    end
movie(M,15)
```

The statement **M=moviein(16)** creates the 2-D structure that stores in each column the data corresponding to a frame at a specific time. The frames themselves are generated within the **for** loop. The **getframe** function returns a pixel image of the image of the different frames. The last command plays the movie *n*-times (15, in this instance).

1.10 Histograms

The most convenient representation for data collected from experiments is in the form of histograms. Typically, you collect data and want to sort it out in different bins; the MATLAB command for this operation is **hist**. But prior to getting to this point, let us introduce some array-related definitions and learn the use of the MATLAB commands that compute them.

Let $\{y_n\}$ be a data set; it can be represented in MATLAB by an array. The largest element of this array is obtained through the command **max(y)**, and the smallest element is obtained through the command **min(y)**.

The mean value of the elements of the array is obtained through the command **mean(y)**, and the standard deviation is obtained through the command **std(y)**.

The definitions of the mean and of the standard deviation are, respectively, given by:

$$\overline{y} = \frac{\sum_{i=1}^{N} y(i)}{N}$$

$$\sigma_y = \sqrt{\frac{N\sum_{i=1}^{N}(y(i))^2 - \left(\sum_{i=1}^{N} y(i)\right)^2}{N(N-1)}}$$

where N is the dimension of the array.

The data (i.e., the array) can be organized into a number of bins (n_b) and exhibited through the command **`[n,y]=hist(y,nb)`**; the array n in the output will be the number of elements in each of the bins.

Example 1.18

Find the mean and the standard deviation and draw the histogram, with 20 bins, for an array whose 10,000 elements are chosen from the MATLAB built-in normal distribution with zero mean and standard deviation 1.

Solution: Edit and execute the following *script M-file:*

```
y=randn(1,10000);
meany=mean(y)
stdy=std(y)
nb=20;
hist(y,nb)
```

You will notice that the results obtained for the mean and the standard deviation vary slightly from the theoretical results. This is due to the finite number of elements chosen for the array and the intrinsic limit in the built-in algorithm used for generating random numbers.

NOTE The MATLAB command for generating an N-elements array of random numbers generated uniformly from the interval [0, 1] is **`rand(1,N)`**.

1.11 Printing and Saving Work in MATLAB

Printing a figure: Use the MATLAB **`print`** function to print a displayed figure directly to your printer. Notice that the printed figure does not take up the entire page. This is because the default orientation of the graph is in portrait mode. To change these settings, try the following commands on an already generated graphic window:

```
orient('landscape') %full horizontal layout
orient('tall')      %full vertical layout
```

Printing a program file (*script M-file*): For both the Mac and PC, open the *M-file* that you want to print. Go to the **File** pull-down menu, and select **Print**.

Saving and loading variables (data): You can use the MATLAB **`save`** function to either save a particular variable or the entire MATLAB workspace. To do this, follow the following example:

```
x=1;y=2;
save 'user volume:x'
save 'user volume:workspace'
```

The first **save** command saved the variable x into a file **x.mat**. You can change the name of the **.mat** file so it does not match the variable name, but that would be confusing. The second command saves all variables (x and y) in the workspace into **workspace.mat**.

To load **x.mat** and **workspace.mat**, enter MATLAB and use the MATLAB load functions; note what you obtain if you entered the following commands:

```
load 'user volume:x'
x
load 'user volume:workspace'
y
```

After loading the variables, you can see a list of all the variables in your workplace if you enter the MATLAB **who** command.

What would you obtain if you had typed and entered the **who** command at this point?

Now, to clear the workspace of some or all variables, use the MATLAB **clear** function.

```
clear x %clears variable x from the workspace
clear   %clears all variables from workspace
```

1.12 MATLAB Commands Review

axis	Sets the axis limits for both 2-D and 3-D plots. Axis supports the arguments equal and square, which makes the current graphs aspect ratio 1.
contour	Plots contour lines of a surface.
clear	Clears all variables from the workspace.
clf	Clears figure.
for	Runs a sequence of commands a given number of times.
getframe	Returns the pixel image of a movie frame.
help	Online help.
hold on(off)	Holds the plot axis with existing graphics on, so that multiple figures can be plotted on the same graph (release the hold of the axes).

if	Conditional evaluation.
length	Gives the length of an array.
load	Loads data or variable values from previous sessions into current MATLAB session.
linspace	Generates an array with a specified number of points between two values.
meshgrid	Makes a 2-D array of coordinate squares suitable for plotting surface meshes.
mesh	Plots a mesh surface of a surface stored in a matrix.
meshc	The same as mesh, but also plots in the same figure the contour plot.
min	Finds the smallest element of an array.
max	Finds the largest element of an array.
mean	Finds the mean of the elements of an array.
moviein	Creates the matrix that contains the frames of an animation.
movie	Plays the movie described by a matrix M.
orient	Orients the current graph to your needs.
plot	Plots points or pairs of arrays on a 2-D graph.
plot3	Plots points or array triples on a 3-D graph.
polar	Plots a polar plot on a polar grid.
pol2cart	Polar to Cartesian conversion.
print	Prints a figure to the default printer.
quit or **exit**	Leave MATLAB program.
rand	Generates an array with elements randomly chosen from the uniform distribution over the interval [0, 1].
randn	Generates an array with elements randomly chosen from the normal distribution function with zero mean and standard deviation 1.
subplot	Partitions the graphics window into sub-windows.
save	Saves MATLAB variables.
std	Finds the standard deviation of the elements of an array.
stem	Plots the data sequence as stems from the *x*-axis terminated with circles for the data value.
view	Views 3-D graphics from different perspectives.
who	Lists all variables in the workspace.
xlabel, ylabel, zlabel, title	Labels the appropriate axes with text and title.
(x>=x1)	Boolean function that is equal to 1 when the condition inside the parenthesis is satisfied, and zero otherwise.