It is usually (but not always) a time-domain operation. Some of the applications are

- removal of unwanted background noise
- removal of interference
- separation of frequency bands
- shaping of the signal spectrum

In some applications, such as voice synthesis, a signal is first analyzed to study its characteristics, which are then used in digital filtering to generate a synthetic voice.

1.2 A BRIEF INTRODUCTION TO MATLAB

MATLAB is an interactive, matrix-based system for scientific and engineering numeric computation and visualization. Its strength lies in the fact that complex numerical problems can be solved easily and in a fraction of the time required by a programming language such as Fortran or C. It is also powerful in the sense that, with its relatively simple programming capability, MATLAB can be easily extended to create new commands and functions.

MATLAB is available in a number of computing environments: PCs running all flavors of Windows, Apple Macs running OS-X, UNIX/Linux workstations, and parallel computers. The basic MATLAB program is further enhanced by the availability of numerous toolboxes (a collection of specialized functions in a specific topic) over the years. The information in this book generally applies to all these environments. In addition to the basic MATLAB product, the Signal Processing toolbox (SP toolbox) is required for this book. The original development of the book was done using the professional version 3.5 running under DOS. The MATLAB scripts and functions described in the book were later extended and made compatible with the present version of MATLAB. Furthermore, through the services of www.cengagebrain.com every effort will be made to preserve this compatibility under future versions of MATLAB.

In this section, we will undertake a brief review of MATLAB. The scope and power of MATLAB go far beyond the few topics discussed in this section. For more detailed tutorial-based discussion, students and readers new to MATLAB should also consult several excellent reference books available in the literature, including [10], [7], and [21]. The information given in all these references, along with the online MATLAB's `help` facility, usually is sufficient to enable readers to use this book. The best approach to become familiar with MATLAB is to open a MATLAB session and experiment with various operators, functions, and commands until

their use and capabilities are understood. Then one can progress to writing simple MATLAB scripts and functions to execute a sequence of instructions to accomplish an analytical goal.

1.2.1 GETTING STARTED

The interaction with MATLAB is through the command window of its graphical user interface (GUI). In the command window, the user types MATLAB instructions, which are executed instantaneously, and the results are displayed in the window. In the MATLAB command window the characters "`>>`" indicate the prompt which is waiting for the user to type a command to be executed. For example,

```
>> command;
```

means an instruction `command` has been issued at the MATLAB prompt. If a semicolon (;) is placed at the end of a command, then all output from that command is suppressed. Multiple commands can be placed on the same line, separated by semicolons ;. Comments are marked by the percent sign (`%`), in which case MATLAB ignores anything to the right of the sign. The comments allow the reader to follow code more easily. The integrated help manual provides help for every command through the fragment

```
>> help command;
```

which will provide information on the inputs, outputs, usage, and functionality of the command. A complete listing of commands sorted by functionality can be obtained by typing `help` at the prompt.

There are three basic elements in MATLAB: numbers, variables, and operators. In addition, punctuation marks (`,`, `;`, `:`, etc.) have special meanings.

Numbers MATLAB is a high-precision numerical engine and can handle all types of numbers, that is, integers, real numbers, complex numbers, among others, with relative ease. For example, the real number 1.23 is represented as simply `1.23` while the real number 4.56×10^7 can be written as `4.56e7`. The imaginary number $\sqrt{-1}$ is denoted either by `1i` or `1j`, although in this book we will use the symbol `1j`. Hence the complex number whose real part is 5 and whose imaginary part is 3 will be written as 5+1j*3. Other constants preassigned by MATLAB are `pi` for π, `inf` for ∞, and `NaN` for not a number (for example, 0/0). These preassigned constants are very important and, to avoid confusion, should not be redefined by users.

Variables In MATLAB, which stands for MATrix LABoratory, the basic variable is a matrix, or an array. Hence, when MATLAB operates on this variable, it operates on all its elements. This is what makes it a powerful and an efficient engine. MATLAB now supports multidimensional arrays; we will discuss only up to two-dimensional arrays of numbers.

1. **Matrix:** A matrix is a two-dimensional set of numbers arranged in rows and columns. Numbers can be real- or complex-valued.
2. **Array:** This is another name for matrix. However, operations on arrays are treated differently from those on matrices. This difference is very important in implementation.

The following are four types of matrices (or arrays):

- **Scalar:** This is a 1×1 matrix or a single number that is denoted by the *variable* symbol, that is, lowercase italic typeface like

$$a = a_{11}$$

- **Column vector:** This is an $(N \times 1)$ matrix or a vertical arrangement of numbers. It is denoted by the *vector* symbol, that is, lowercase bold typeface like

$$\mathbf{x} = [x_{i1}]_{i:1,\ldots,N} = \begin{bmatrix} x_{11} \\ x_{21} \\ \vdots \\ x_{N1} \end{bmatrix}$$

A typical vector in linear algebra is denoted by the column vector.

- **Row vector:** This is a $(1 \times M)$ matrix or a horizontal arrangement of numbers. It is also denoted by the vector symbol, that is,

$$\mathbf{y} = [y_{1j}]_{j=1,\ldots,M} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1M} \end{bmatrix}$$

A one-dimensional discrete-time signal is typically represented by an array as a row vector.

- **General matrix:** This is the most general case of an $(N \times M)$ matrix and is denoted by the matrix symbol, that is, uppercase bold typeface like

$$\mathbf{A} = [a_{ij}]_{i=1,\ldots,N;j=1,\ldots,m} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NM} \end{bmatrix}$$

This arrangement is typically used for two-dimensional discrete-time signals or images.

MATLAB does not distinguish between an array and a matrix except for operations. The following assignments denote indicated matrix types in MATLAB:

`a = [3]` is a scalar,
`x = [1,2,3]` is a row vector,
`y = [1;2;3]` is a column vector, and
`A = [1,2,3;4,5,6]` is a matrix.

MATLAB provides many useful functions to create special matrices. These include `zeros(M,N)` for creating a matrix of all zeros, `ones(M,N)` for creating matrix of all ones, `eye(N)` for creating an $N \times N$ identity matrix, etc. Consult MATLAB's help manual for a complete list.

Operators MATLAB provides several arithmetic and logical operators, some of which follow. For a complete list, MATLAB's help manual should be consulted.

`=` assignment	`==` equality
`+` addition	`-` subtraction or minus
`*` multiplication	`.*` array multiplication
`^` power	`.^` array power
`/` division	`./` array division
`<>` relational operators	`&` logical AND
`|` logical OR	`~` logical NOT
`'` transpose	`.'` array transpose

We now provide a more detailed explanation on some of these operators.

1.2.2 MATRIX OPERATIONS

Following are the most useful and important operations on matrices.

- **Matrix addition and subtraction:** These are straightforward operations that are also used for array addition and subtraction. Care must be taken that the two matrix operands be *exactly* the same size.
- **Matrix conjugation:** This operation is meaningful only for complex-valued matrices. It produces a matrix in which all imaginary parts are negated. It is denoted by $\mathbf{A}^*$ in analysis and by `conj(A)` in MATLAB.
- **Matrix transposition:** This is an operation in which every row (column) is turned into column (row). Let $\mathbf{X}$ be an $(N \times M)$ matrix. Then

$$\mathbf{X}^{'} = [x_{ji}]\,; \quad j = 1, \ldots, M,\ i = 1, \ldots, N$$

is an $(M \times N)$ matrix. In MATLAB, this operation has one additional feature. If the matrix is real-valued, then the operation produces the

usual transposition. However, if the matrix is complex-valued, then the operation produces a complex-conjugate transposition. To obtain just the transposition, we use the array operation of conjugation, that is, `A.'` will do just the transposition.

- **Multiplication by a scalar:** This is a simple straightforward operation in which each element of a matrix is scaled by a constant, that is,

$$ab \Rightarrow \texttt{a*b} \text{ (scalar)}$$

$$a\mathbf{x} \Rightarrow \texttt{a*x} \text{ (vector or array)}$$

$$a\mathbf{X} \Rightarrow \texttt{a*X} \text{ (matrix)}$$

  This operation is also valid for an array scaling by a constant.

- **Vector-vector multiplication:** In this operation, one has to be careful about matrix dimensions to avoid invalid results. The operation produces either a scalar or a matrix. Let $\mathbf{x}$ be an $(N \times 1)$ and $\mathbf{y}$ be a $(1 \times M)$ vectors. Then

$$\texttt{x} * \texttt{y} \Rightarrow \mathbf{xy} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix} = \begin{bmatrix} x_1 y_1 & \cdots & x_1 y_M \\ \vdots & \ddots & \vdots \\ x_N y_1 & \cdots & x_N y_M \end{bmatrix}$$

  produces a matrix. If $M = N$, then

$$\texttt{y} * \texttt{x} \Rightarrow \mathbf{yx} = \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} = x_1 y_1 + \cdots + x_M y_M$$

- **Matrix-vector multiplication:** If the matrix and the vector are compatible (i.e., the number of matrix-columns is equal to the vector-rows), then this operation produces a column vector:

$$\texttt{y = A*x} \Rightarrow \mathbf{y} = \mathbf{Ax} = \begin{bmatrix} a_{11} & \cdots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NM} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

- **Matrix-matrix multiplication:** Finally, if two matrices are compatible, then their product is well-defined. The result is also a matrix with the number of rows equal to that of the first matrix and the number of columns equal to that of the second matrix. Note that the order in matrix multiplication is very important.

Array Operations These operations treat matrices as arrays. They are also known as *dot operations* because the arithmetic operators are prefixed by a dot (`.`), that is, `.*`, `./`, or `.^`.

- **Array multiplication:** This is an element by element multiplication operation. For it to be a valid operation, both arrays must be the same size. Thus we have

  `x.*y` → 1D array

  `X.*Y` → 2D array

- **Array exponentiation:** In this operation, a scalar (real- or complex-valued) is raised to the power equal to every element in an array, that is,

$$\texttt{a.\^{}x} \equiv \begin{bmatrix} a^{x_1} \\ a^{x_2} \\ \vdots \\ a^{x_N} \end{bmatrix}$$

  is an $(N \times 1)$ array, whereas

$$\texttt{a.\^{}X} \equiv \begin{bmatrix} a^{x_{11}} & a^{x_{12}} & \cdots & a^{x_{1M}} \\ a^{x_{21}} & a^{x_{22}} & \cdots & a^{x_{2M}} \\ \vdots & \vdots & \ddots & \vdots \\ a^{x_{N1}} & a^{x_{N2}} & \cdots & a^{x_{NM}} \end{bmatrix}$$

  is an $(N \times M)$ array.
- **Array transposition:** As explained, the operation `A.'` produces transposition of real- or complex-valued array `A`.

Indexing Operations MATLAB provides very useful and powerful array indexing operations using operator `:`. It can be used to generate sequences of numbers as well as to access certain row/column elements of a matrix. Using the fragment `x = [a:b:c]`, we can generate numbers from `a` to `c` in `b` increments. If `b` is positive (negative) then, we get increasing (decreasing) values in the sequence `x`.

The fragment `x(a:b:c)` accesses elements of `x` beginning with index `a` in steps of `b` and ending at `c`. Care must be taken to use integer values of indexing elements. Similarly, the `:` operator can be used to extract a submatrix from a matrix. For example, `B = A(2:4,3:6)` extracts a 3×4 submatrix starting at row 2 and column 3.

Another use of the `:` operator is in forming column vectors from row vectors or matrices. When used on the right-hand side of the equality (`=`) operator, the fragment `x=A(:)` forms a long column vector `x` of elements

of `A` by concatenating its columns. Similarly, `x=A(:,3)` forms a vector `x` from the third column of `A`. However, when used on the right-hand side of the `=` operator, the fragment `A(:)=x` reformats elements in `x` into a predefined size of `A`.

Control-Flow MATLAB provides a variety of commands that allow us to control the flow of commands in a program. The most common construct is the `if-elseif-else` structure. With these commands, we can allow different blocks of code to be executed depending on some condition. The format of this construct is

```
 if condition1
   command1
elseif condition2
   command2
else
   command3
end
```

which executes statements in `command1` if `condition-1` is satisfied; otherwise statements in `command2` if `condition-2` is satisfied, or finally statements in `command3`.

Another common control flow construct is the `for..end` loop. It is simply an iteration loop that tells the computer to repeat some task a given number of times. The format of a `for..end` loop is

```
 for index = values
   program statements
         :
end
```

Although `for..end` loops are useful for processing data inside of arrays by using the iteration variable as an index into the array, whenever possible the user should try to use MATLAB's whole array mathematics. This will result in shorter programs and more efficient code. In some situations the use of the `for..end` loop is unavoidable. The following example illustrates these concepts.

□ **EXAMPLE 1.1** Consider the following sum of sinusoidal functions.

$$x(t) = \sin(2\pi t) + \tfrac{1}{3}\sin(6\pi t) + \tfrac{1}{5}\sin(10\pi t) = \sum_{k=1}^{3} \frac{1}{k}\sin(2\pi k t), \qquad 0 \le t \le 1$$

Using MATLAB, we want to generate samples of $x(t)$ at time instances `0:0.01:1`. We will discuss three approaches.

Approach 1

Here we will consider a typical C or Fortran approach, that is, we will use two `for..end` loops, one each on `t` and `k`. This is the most inefficient approach in MATLAB, but possible.

```
>> t = 0:0.01:1; N = length(t); xt = zeros(1,N);
>> for n = 1:N
>>     temp = 0;
>>     for k = 1:3
>>         temp = temp + (1/k)*sin(2*pi*k*t(n));
>>     end
>>     xt(n) = temp;
>> end
```

Approach 2

In this approach, we will compute each sinusoidal component in one step as a vector, using the time vector `t = 0:0.01:1` and then add all components using one `for..end` loop.

```
>> t = 0:0.01:1; xt = zeros(1,length(t));
>> for k = 1:3
>>    xt = xt + (1/k)*sin(2*pi*k*t);
>> end
```

Clearly, this is a better approach with fewer lines of code than the first one.

Approach 3

In this approach, we will use matrix-vector multiplication, in which MATLAB is very efficient. For the purpose of demonstration, consider only four values for $t = [t_1, t_2, t_3, t_4]$. Then

$$x(t_1) = \sin(2\pi t_1) + \tfrac{1}{3}\sin(2\pi 3t_1) + \tfrac{1}{5}\sin(2\pi 5t_1)$$

$$x(t_2) = \sin(2\pi t_2) + \tfrac{1}{3}\sin(2\pi 3t_2) + \tfrac{1}{5}\sin(2\pi 5t_2)$$

$$x(t_3) = \sin(2\pi t_3) + \tfrac{1}{3}\sin(2\pi 3t_3) + \tfrac{1}{5}\sin(2\pi 5t_3)$$

$$x(t_4) = \sin(2\pi t_4) + \tfrac{1}{3}\sin(2\pi 3t_4) + \tfrac{1}{5}\sin(2\pi 5t_4)$$

which can be written in matrix form as

$$\begin{bmatrix} x(t_1) \\ x(t_2) \\ x(t_3) \\ x(t_4) \end{bmatrix} = \begin{bmatrix} \sin(2\pi t_1) & \sin(2\pi 3t_1) & \sin(2\pi 5t_1) \\ \sin(2\pi t_2) & \sin(2\pi 3t_2) & \sin(2\pi 5t_2) \\ \sin(2\pi t_3) & \sin(2\pi 3t_3) & \sin(2\pi 5t_3) \\ \sin(2\pi t_4) & \sin(2\pi 3t_4) & \sin(2\pi 5t_4) \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1}{3} \\ \frac{1}{5} \end{bmatrix}$$

$$= \sin\left(2\pi \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} \begin{bmatrix} 1 & 3 & 5 \end{bmatrix} \right) \begin{bmatrix} 1 \\ \frac{1}{3} \\ \frac{1}{5} \end{bmatrix}$$

or after taking transposition

$$\begin{bmatrix} x(t_1) & x(t_2) & x(t_3) & x(t_4) \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{5} \end{bmatrix} \sin\left(2\pi \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \begin{bmatrix} t_1 & t_2 & t_3 & t_4 \end{bmatrix} \right)$$

Thus the MATLAB code is

```
>> t = 0:0.01:1; k = 1:3;
>> xt = (1./k)*sin(2*pi*k'*t);
```

Note the use of the array division (`1./k`) to generate a row vector and matrix multiplications to implement all other operations. This is the most compact code and the most efficient execution in MATLAB, especially when the number of sinusoidal terms is very large.

1.2.3 SCRIPTS AND FUNCTIONS

MATLAB is convenient in the interactive command mode if we want to execute few lines of code. But it is not efficient if we want to write code of several lines that we want to run repeatedly or if we want to use the code in several programs with different variable values. MATLAB provides two constructs for this purpose.

Scripts The first construct can be accomplished by using the so-called block mode of operation. In MATLAB, this mode is implemented using a *script* file called an m-file (with an extension `.m`), which is only a text file that contains each line of the file as though you typed them at the command prompt. These scripts are created using MATLAB's built-in editor, which also provides for context-sensitive colors and indents for making fewer mistakes and for easy reading. The script is executed by typing the name of the script at the command prompt. The script file must be in the current directory on in the directory of the `path` environment. As an example, consider the sinusoidal function in Example 1.1. A general form of this function is

$$x(t) = \sum_{k=1}^{K} c_k \sin(2\pi k t) \tag{1.1}$$

If we want to experiment with different values of the coefficients c_k and/or the number of terms K, then we should create a script file. To implement the third approach in Example 1.1, we can write a script file

```
% Script file to implement (1.1)
t = 0:0.01:1; k = 1:2:5; ck = 1./k;
xt = ck * sin(2*pi*k'*t);
```

Now we can experiment with different values.

Functions The second construct of creating a block of code is through subroutines. These are called *functions*, which also allow us to extend the capabilities of MATLAB. In fact a major portion of MATLAB is assembled using function files in several categories and using special collections called *toolboxes*. Functions are also m-files (with extension `.m`). A major difference between script and function files is that the first executable line in a function file begins with the keyword `function` followed by an output-input variable declaration. As an example, consider the computation of the $x(t)$ function in Example 1.1 with an arbitrary number of sinusoidal terms, which we will implement as a function stored as m-file `sinsum.m`.

```
function xt = sinsum(t,ck)
% Computes sum of sinusoidal terms of the form in (1.1)
% x = sinsum(t,ck)
%
K = length(ck); k = 1:K;
ck = ck(:)'; t = t(:)';
xt = ck * sin(2*pi*k'*t);
```

The vectors `t` and `ck` should be assigned prior to using the `sinsum` function. Note that `ck(:)'` and `t(:)'` use indexing and transposition operations to force them to be row vectors. Also note the comments immediately following the `function` declaration, which are used by the `help sinsum` command. Sufficient information should be given there for the user to understand what the function is supposed to do.

1.2.4 PLOTTING

One of the most powerful features of MATLAB for signal and data analysis is its graphical data plotting. MATLAB provides several types of plots, starting with simple two-dimensional (2D) graphs to complex, higher-dimensional plots with full-color capability. We will examine only the 2D plotting, which is the plotting of one vector versus another in a 2D coordinate system. The basic plotting command is the `plot(t,x)` command, which generates a plot of `x` values versus `t` values in a separate figure window. The arrays `t` and `x` should be the same length and orientation. Optionally, some additional formatting keywords can also be provided in the `plot` function. The commands `xlabel` and `ylabel` are used to add text to the axis, and the command `title` is used to provide a title on the top of the graph. When plotting data, one should get into the habit of always labeling the axis and providing a title. Almost all aspects of a plot (style, size, color, etc.) can be changed by appropriate commands embedded in the program or directly through the GUI.

The following set of commands creates a list of sample points, evaluates the sine function at those points, and then generates a plot of a simple sinusoidal wave, putting axis labels and title on the plot.

```
>> t = 0:0.01:2; % sample points from 0 to 2 in steps of 0.01
>> x = sin(2*pi*t); % Evaluate sin(2 pi t)
>> plot(t,x,'b'); % Create plot with blue line
>> xlabel('t in sec'); ylabel('x(t)'); % Label axis
>> title('Plot of sin(2\pi t)'); % Title plot
```

The resulting plot is shown in Figure 1.1.

For plotting a set of discrete numbers (or discrete-time signals), we will use the `stem` command which displays data values as a stem, that is, a small circle at the end of a line connecting it to the horizontal axis. The circle can be open (default) or filled (using the option `'filled'`). Using Handle Graphics (MATLAB's extensive manipulation of graphics primitives), we can resize circle markers. The following set of commands displays a discrete-time sine function using these constructs.

```
>> n = 0:1:40; % sample index from 0 to 20
>> x = sin(0.1*pi*n); % Evaluate sin(0.2 pi n)
>> Hs = stem(n,x,'b','filled'); % Stem-plot with handle Hs
>> set(Hs,'markersize',4); % Change circle size
>> xlabel('n'); ylabel('x(n)'); % Label axis
>> title('Stem Plot of sin(0.2 pi n)'); % Title plot
```

The resulting plot is shown in Figure 1.2.

MATLAB provides an ability to display more than one graph in the same figure window. By means of the `hold on` command, several graphs can be plotted on the same set of axes. The `hold off` command stops the simultaneous plotting. The following MATLAB fragment (Figure 1.3)

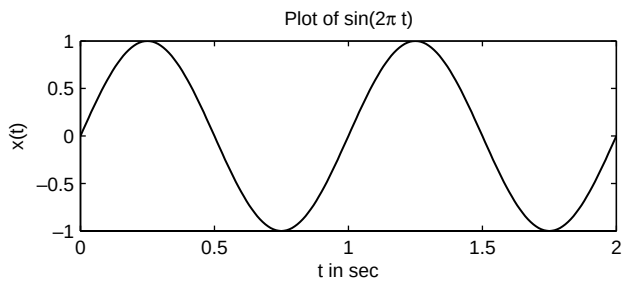

FIGURE 1.1 *Plot of the* $\sin(2\pi t)$ *function*

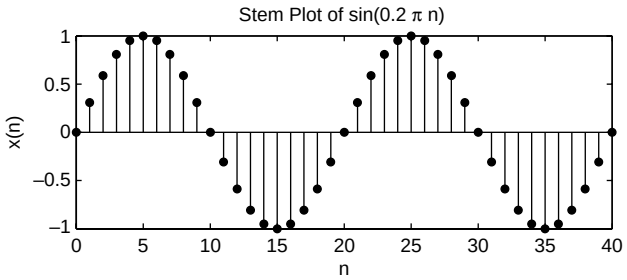

FIGURE 1.2 *Plot of the sin(0.2π n) sequence*

displays graphs in Figures 1.1 and 1.2 as one plot, depicting a "sampling" operation that we will study later.

```
>> plot(t,xt,'b'); hold on; % Create plot with blue line
>> Hs = stem(n*0.05,xn,'b','filled'); % Stem-plot with handle Hs
>> set(Hs,'markersize',4); hold off; % Change circle size
```

Another approach is to use the `subplot` command, which displays several graphs in each individual set of axes arranged in a grid, using the parameters in the `subplot` command. The following fragment (Figure 1.4) displays graphs in Figure 1.1 and 1.2 as two separate plots in two rows.

```
. . .
>> subplot(2,1,1); % Two rows, one column, first plot
>> plot(t,x,'b'); % Create plot with blue line
. . .
>> subplot(2,1,2); % Two rows, one column, second plot
>> Hs = stem(n,x,'b','filled'); % Stem-plot with handle Hs
. . .
```

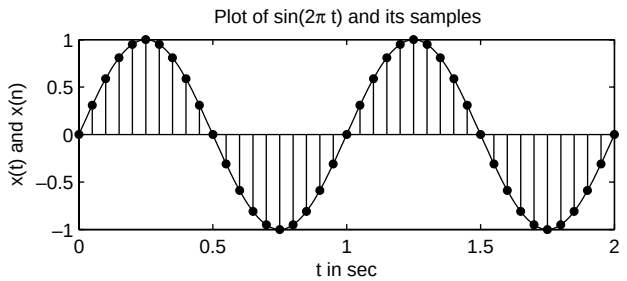

FIGURE 1.3 *Simultaneous plots of $x(t)$ and $x(n)$*

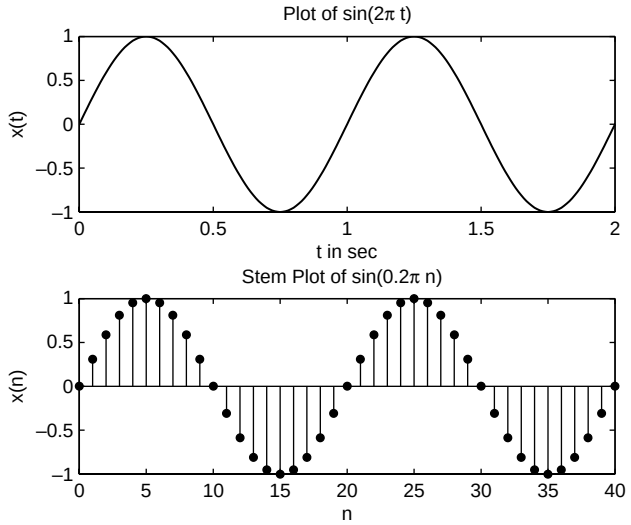

FIGURE 1.4 *Plots of $x(t)$ and $x(n)$ in two rows*

The plotting environment provided by MATLAB is very rich in its complexity and usefulness. It is made even richer using the handle-graphics constructs. Therefore, readers are strongly recommended to consult MATLAB's manuals on plotting. Many of these constructs will be used throughout this book.

In this brief review, we have barely made a dent in the enormous capabilities and functionalities in MATLAB. Using its basic integrated help system, detailed help browser, and tutorials, it is possible to acquire sufficient skills in MATLAB in a reasonable amount of time.

1.3 APPLICATIONS OF DIGITAL SIGNAL PROCESSING

The field of DSP has matured considerably over the last several decades and now is at the core of many diverse applications and products. These include

- speech/audio (speech recognition/synthesis, digital audio, equalization, etc.),
- image/video (enhancement, coding for storage and transmission, robotic vision, animation, etc.),
- military/space (radar processing, secure communication, missile guidance, sonar processing, etc.),
- biomedical/health care (scanners, ECG analysis, X-ray analysis, EEG brain mappers, etc.)