Chapter 1

Signals and Systems

The basic concepts of signals and systems arise in a variety of contexts, from engineering design to financial analysis. In this chapter, you will learn how to represent, manipulate, and analyze basic signals and systems in MATLAB. The first section of this chapter, Tutorial 1.1, covers some of the fundamental tools used to construct signals in MATLAB. This tutorial is meant to be a supplement to, but not a substitute for, the tutorial given in *The MATLAB User's Guide*. If you have not already done so, you are strongly encouraged to work through this tutorial or the demos found in the MATLAB software (e.g. `intro`) before beginning this chapter. While not all of the MATLAB functions introduced in these tutorials are needed for the exercises of this chapter, most will be used at some point in this book.

Complex exponential signals are used frequently for signals and systems analysis, in part because complex exponential signals form the building blocks of large classes of signals. Exercise 1.2 covers the MATLAB functions required for generating and plotting discrete-time sinusoidal signals, which are equal to the sum of two discrete-time complex exponential signals, i.e.,

$$\cos(\omega n) = \frac{1}{2}\left(e^{i\omega n} + e^{-i\omega n}\right), \tag{1.1}$$

$$\sin(\omega n) = \frac{1}{2i}\left(e^{i\omega n} - e^{-i\omega n}\right). \tag{1.2}$$

Exercise 1.3 shows how to plot discrete-time signals $x[n]$ after transformations of the independent variable n. The next two exercises cover system representations in MATLAB. For Exercise 1.4, you must demonstrate your understanding of basic system properties like linearity and time-invariance. For Exercise 1.5, you must implement a system described by a first-order difference equation.

Several of the exercises in this chapter use the Symbolic Math Toolbox to study basic signals and systems. In Exercise 1.6, you will construct symbolic expressions for continuous-time complex exponential signals, which have the form e^{st} for some complex number s. (Note that both i and j will be used in this book to represent the imaginary number $\sqrt{-1}$.) Exercise 1.7 uses the Symbolic Math Toolbox to implement transformations on the time-index of continuous-time signals. For Exercise 1.8, you must create analytic expressions for the energy of periodic signals and relate energy to time-averaged power.

From "Computer Explorations in Signals and Systems using Matlab, 2nd Ed."
Buck, Daniel, and Singer; Prentice Hall

■ 1.1 Tutorial: Basic MATLAB Functions for Representing Signals

In this tutorial, you will learn how to use several MATLAB functions that will frequently be used to construct and manipulate signals in this book. If you have not already done so, you are strongly encouraged to work through the tutorials in *The MATLAB User's Guide* or in the MATLAB software. This tutorial is not meant to replace those tutorials, but rather to illustrate how some of the functions described there can be used for representing and working with signals. Although there are no problems to be worked in this tutorial, you should duplicate all the examples in MATLAB to give yourself practice with the commands. In general, signals will be represented by a row or column vector, depending on the context. All vectors represented in MATLAB are indexed starting with 1, i.e., `y(1)` is the first element of the vector `y`. If these indices do not correspond to those in your application, you can create an additional index vector to properly keep track of the signal index. For example, to represent the discrete-time signal

$$x[n] = \begin{cases} 2n, & -3 \le n \le 3, \\ 0, & \text{otherwise}, \end{cases}$$

you could first use the colon operator to define the index vector for the nonzero samples of $x[n]$, and then define the vector `x` to contain the values of the signal at each of these time indices:

```
>> n = [-3:3];
>> x = 2*n;
```

Note that we have used semicolons at the end of each command to suppress unnecessary MATLAB echoing. For instance, without the semicolon you would get

```
>> n = [-3:3]
n =
    -3    -2    -1     0     1     2     3
```

You can plot this signal by typing `stem(n,x)`. If you want to examine the signal over a wider range of indices, you will need to extend both `n` and `x`. For instance, if you want to plot the signal over the range $-5 \le n \le 5$, you can extend the index vector `n`, then add additional elements to `x` for these new samples:

```
>> n = [-5:5];
>> x = [0 0 x 0 0 ];
```

If you want to greatly extend the range of the signal, you may find it helpful to use the function `zeros`. For instance, if you wanted to include the region $-100 \le n \le 100$, after you had already extended `x` to include $-5 \le n \le 5$ as shown above, you could type

```
>> n = [-100:100];
>> x = [zeros(1,95) x zeros(1,95)];
```

Suppose you want to define $x_1[n]$ to be the discrete-time unit impulse function and $x_2[n]$ to be a time-advanced version of $x_1[n]$, i.e., $x_1[n] = \delta[n]$ and $x_2[n] = \delta[n+2]$. You could represent these signals in MATLAB by typing

```
>> nx1 = [0:10];
>> x1 = [1 zeros(1,10)];
>> nx2 = [-5:5];
>> x2 = [zeros(1,3) 1 zeros(1,7)];
```

You could then plot these signals by `stem(nx1,x1)` and `stem(nx2,x2)`. If you did not define the index vectors, and simply typed `stem(x1)` and `stem(x2)`, you would make plots of the signals $\delta[n-1]$ and $\delta[n-4]$ and not of the desired signals. The index vector will also be useful for keeping track of the time origin of a vector when you work on more advanced exercises in later chapters.

We will explore two methods for representing continuous-time signals in MATLAB. One method is to use the Symbolic Math Toolbox. The exercises in this book which use the Symbolic Math Toolbox are marked by the symbol Ⓢ at the end of the exercise title. You can also represent continuous-time signals with vectors containing closely spaced samples of the signals in time. The projects in early chapters that represent continuous-time signals by closely spaced samples will always explicitly specify the time spacing to use to guarantee that the signal is accurately represented. In Chapter 7, you will explore the issues involved with representing a continuous-time signal by discrete-time samples. Vectors of closely spaced time indices can be created in a number of ways. Two simple methods are to use the colon operator with the optional step argument, and to use the `linspace` function. For instance, if you wanted to create a vector that covered the interval $-5 \leq t \leq 5$ in steps of 0.1 seconds, you could either use `t=[-5:0.1:5]` or `t=linspace(-5,5,101)`.

Sinusoids and complex exponentials are important signals for the study of linear systems. MATLAB provides several functions that are useful for defining such signals, especially if you have already defined either a continuous-time or discrete-time index vector. For instance, if you wanted to form a vector to represent $x(t) = \sin(\pi t/4)$ for $-5 \leq t \leq 5$, you could use the vector `t` defined in the previous paragraph and type `x=sin(pi*t/4)`. Note that when the argument to `sin` (or many other MATLAB functions, such as `cos` and `exp`) is a vector, the function returns a vector of the same size where each element of the output vector is the function applied to the corresponding element of the input vector.

You can use the `plot` command to plot your approximation to the continuous-time signal $x(t)$. Unlike `stem`, `plot` connects adjacent elements with a straight line, so that when the time index is finely sampled, the straight lines are a close approximation to a plot of the original continuous-time signal. For this example, you can generate such a plot by typing `plot(t,x)`. In general, you will want to use `stem` to plot short discrete-time sequences, and `plot` for sampled approximations of continuous-time signals or for very long discrete-time signals where the number of stems grows unwieldy.

Discrete-time sinusoids and complex exponentials can also be generated using `cos`, `sin`, and `exp`. For instance, to represent the discrete-time signal $x[n] = e^{j(\pi/8)n}$ for $0 \leq n \leq 32$, you would type

```
>> n = [0:32];
>> x = exp(j*(pi/8)*n);
```

The vector x now contains the complex values of the signal $x[n]$ over the interval $0 \leq n \leq 32$. To plot complex signals, you must plot their real and imaginary parts, or magnitude and angle, separately. The MATLAB functions `real`, `imag`, `abs`, and `angle` compute these functions of a complex vector on a term-by-term basis. You can plot each of these functions of this complex signal by typing

```
>> stem(n,real(x))
>> stem(n,imag(x))
>> stem(n,abs(x))
>> stem(n,angle(x))
```

For the last example, note that the value returned by `angle` is the phase of the complex number in radians. To convert to degrees, type `stem(n,angle(x)*(180/pi))`.
MATLAB also allows you to add, subtract, multiply, divide, scale and exponentiate signals. As long as the vectors representing the signals have the same time-origins and the same number of elements, e.g.,

```
>> x1 = sin((pi/4)*[0:15]);
>> x2 = cos((pi/7)*[0:15]);
```

you can perform the following term-by-term operations:

```
>> y1 = x1+x2;
>> y2 = x1-x2;
>> y3 = x1.*x2;
>> y4 = x1./x2;
>> y5 = 2*x1;
>> y6 = x1.^3;
```

Note that for multiplying, dividing and exponentiating on a term-by-term basis, you must precede the operator with a period, i.e., use the `.*` function instead of just `*` for term-by-term multiplication. MATLAB interprets the `*` operator without a period to be the matrix multiplication operator, not term-by-term multiplication. For example, if you try to multiply x1 and x2 using `*`, you will receive the following error message:

```
>> x1*x2
??? Error using ==> *
Inner matrix dimensions must agree.
```

because matrix multiplication requires that the number of columns of the first argument be equal to the number of rows of the second argument, which is not true for the two 1×5 vectors x1 and x2. You must also be careful to use `./` and `.^` when operating on vectors

term-by-term, since / and ^ are matrix operations.
MATLAB also includes several commands to help you label plots appropriately, as well as to print them out. The `title` command places its argument over the current plot as the title. The commands `xlabel` and `ylabel` allow you to label the axes of your graph, making it clear what has been plotted. Every plot or graph you generate should have a title, as well as labels for both axes. For example, consider again a plot of the following signal and index vector:

```
>> n = [0:32];
>> x = exp(j*(pi/8)*n);
>> stem(n,angle(x))
```

You could label your graph by typing

```
>> title('Phase of exp(j*(pi/8)*n)')
>> xlabel('n (samples)')
>> ylabel('Phase of x[n] (radians)')
```

The `print` command allows you to print out the current plot. You should type `help print` to understand how it works on your system, as it will vary slightly depending on the operating system and configuration of the computer you are using.
Another important feature of MATLAB is the ability to write M-files. There are two types of M-files: functions and command scripts . A command script is a text file of MATLAB commands whose filename ends in `.m` in the current working directory or elsewhere on your MATLABPATH. If you type the name of this file (without the `.m`), the commands contained in the file will be executed. Using these scripts will make it much easier for you to do the exercises in this book. Many exercises will require you to process several signals in a similar or identical way. If you do not use scripts, you will have to retype all the commands anew. However, if you did the first problem using a script, you can process all the subsequent signals in that exercise by copying the script file and editing it to process the new signal.
For example, suppose you had the following script file `prob1.m` to plot the discrete-time signal $\cos(\pi n/4)$ and compute its mean over the interval $0 \leq n \leq 16$:

```
% prob1.m
n = [0:16];
x1 = cos(pi*n/4);
y1 = mean(x1);
stem(n,x1)
title('x1 = cos(pi*n/4)')
xlabel('n (samples)')
ylabel('x1[n]')
```

If you then wanted to do the same for $x_2[n] = \sin(\pi n/4)$, you could copy `prob1.m` to `prob2.m`, then edit it slightly to get

```
% prob2.m
n = [0:16];
x2 = sin(pi*n/4);
y2 = mean(x2);
stem(n,x2)
title('x = sin(pi*n/4)')
xlabel('n (samples)')
ylabel('x2[n]')
```

You can then type `prob2` to run these commands and generate the desired plot and compute the average value of the new signal. Instead of retyping all 7 lines, you need only edit about a dozen characters. We strongly encourage you to use scripts in working the problems in this book, with a separate script for each exercise, or even each problem. Scripts also make debugging your work much easier, as you can fix one mistake and then easily rerun the modified sequence of commands. Finally, when you complete an exercise, it is easy to print out your script file and hand it in as a record of your work.

An M-file implementing a function is a text file with a title ending in `.m` whose first word is `function`. The rest of the first line of the file specifies the names of the input and output arguments of the function. For example, the following M-file implements a function called `foo` which accepts an input `x` and returns `y` and `z`, which are equal to `2*x` and `(5/9)*(x-32)`, respectively:

```
function [y,z] = foo(x)
% [y,z] = foo(x) accepts a numerical argument x and
% returns two arguments y and z, where y is 2*x
% and z is (5/9)*(x-32)
y = 2*x;
z = (5/9)*(x-32);
```

Two sample calls to `foo` are shown below:

```
>> [y,z] = foo(-40)
y =
   -80
z =
   -40

>> [y,z] = foo(212)
y =
   424
z =
   100
```

The commands described in this tutorial are by no means the complete set you will need to do the exercises in this book, but instead are meant to get you started using MATLAB. Future exercises in this book will assume that you are comfortable using the commands discussed here, and that you are also able to learn about other basic mathematical commands

in MATLAB by using either the manual or the `help` function. Specialized functions for signal processing will often be described in their own tutorials in later chapters. Again, if you have not already done so, you should work through the general tutorial in the MATLAB manual so you are familiar with the functions available in MATLAB.

■ 1.2 Discrete-Time Sinusoidal Signals

Discrete-time complex exponentials play an important role in the analysis of discrete-time signals and systems. A discrete-time complex exponential has the form α^n, where α is a complex scalar. The discrete-time sine and cosine signals can be built from complex exponential signals by setting $\alpha = e^{\pm i\omega}$, namely,

$$\cos(\omega n) = \frac{1}{2}\left(e^{i\omega n} + e^{-i\omega n}\right), \tag{1.3}$$

$$\sin(\omega n) = \frac{1}{2i}\left(e^{i\omega n} - e^{-i\omega n}\right). \tag{1.4}$$

In this exercise, you will create and analyze a number of discrete-time sinusoids. There are many similarities between continuous-time and discrete-time sinusoids, as follows from a simple comparison of Eqs. (1.3)-(1.4) and Eqs. (1.7)-(1.8). However, you will also examine some of the important differences between sinusoids in continuous and discrete time in this exercise.

Basic Problems

(a). Consider the discrete-time signal

$$x_M[n] = \sin\left(\frac{2\pi M n}{N}\right),$$

and assume $N = 12$. For $M = 4$, 5, 7, and 10, plot $x_M[n]$ on the interval $0 \le n \le 2N - 1$. Use `stem` to create your plots, and be sure to appropriately label your axes. What is the fundamental period of each signal? In general, how can the fundamental period be determined from arbitrary integer values of M and N? Be sure to consider the case in which $M > N$.

(b). Consider the signal

$$x_k[n] = \sin(\omega_k n),$$

where $\omega_k = 2\pi k/5$. For $x_k[n]$ given by $k = 1$, 2, 4, and 6, use `stem` to plot each signal on the interval $0 \le n \le 9$. All of the signals should be plotted with separate axes in the same figure using `subplot`. How many unique signals have you plotted? If two signals are identical, explain how different values of ω_k can yield the same signal.

(c). Now consider the following three signals:

$$x_1[n] = \cos\left(\frac{2\pi n}{N}\right) + 2\cos\left(\frac{3\pi n}{N}\right),$$
$$x_2[n] = 2\cos\left(\frac{2n}{N}\right) + \cos\left(\frac{3n}{N}\right),$$
$$x_3[n] = \cos\left(\frac{2\pi n}{N}\right) + 3\sin\left(\frac{5\pi n}{2N}\right).$$

Assume $N = 6$ for each signal. Determine whether or not each signal is periodic. If a signal is periodic, plot the signal for two periods, starting at $n = 0$. If the signal is not periodic, plot the signal for $0 \leq n \leq 4N$ and explain why it is not periodic. Remember to use `stem` and to appropriately label your axes.

Intermediate Problems

(d). Plot each of the following signals on the interval $0 \leq n \leq 31$:

$$x_1[n] = \sin\left(\frac{\pi n}{4}\right)\cos\left(\frac{\pi n}{4}\right),$$
$$x_2[n] = \cos^2\left(\frac{\pi n}{4}\right),$$
$$x_3[n] = \sin\left(\frac{\pi n}{4}\right)\cos\left(\frac{\pi n}{8}\right).$$

What is the fundamental period of each signal? For each of these three signals, how could you have determined the fundamental period without relying upon MATLAB?

(e). Consider the signals you plotted in Parts (c) and (d). Is the addition of two periodic signals necessarily periodic? Is the multiplication of two periodic signals necessarily periodic? Clearly explain your answers.

■ 1.3 Transformations of the Time Index for Discrete-Time Signals

In this exercise you will examine how to use MATLAB to represent discrete-time signals. In addition, you will explore the effect of simple transformations of the independent variable, such as delaying the signal or reversing its time axis. These rudimentary transformations of the independent variable will occur frequently in studying signals and systems, so becoming comfortable and confident with them now will benefit you in studying more advanced topics.

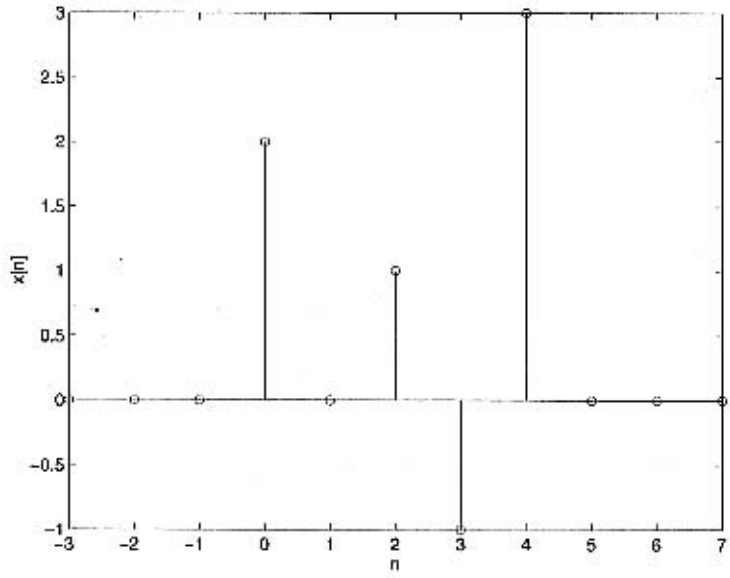

Figure 1.1. Discrete-time signal $x[n]$.

Basic Problems

(a). Define a MATLAB vector `nx` to be the time indices $-3 \leq n \leq 7$ and the MATLAB vector `x` to be the values of the signal $x[n]$ at those samples, where $x[n]$ is given by

$$x[n] = \begin{cases} 2, & n = 0, \\ 1, & n = 2, \\ -1, & n = 3, \\ 3, & n = 4, \\ 0, & \text{otherwise}. \end{cases}$$

If you have defined these vectors correctly, you should be able to plot this discrete-time sequence by typing `stem(nx,x)`. The resulting plot should match the plot shown in Figure 1.1.

(b). For this part, you will define MATLAB vectors `y1` through `y4` to represent the following discrete-time signals:

$$y_1[n] = x[n-2],$$
$$y_2[n] = x[n+1],$$
$$y_3[n] = x[-n],$$
$$y_4[n] = x[-n+1].$$

To do this, you should define `y1` through `y4` to be equal to `x`. The key is to define correctly the corresponding index vectors `ny1` through `ny4`. First, you should figure

out how the index of a given sample of $x[n]$ changes when transforming to $y_i[n]$. The index vectors need not span the same set of indices as nx, but they should all be at least 11 samples long and include the indices of all nonzero samples of the associated signal.

(c). Generate plots of $y_1[n]$ through $y_4[n]$ using **stem**. Based on your plots, state how each signal is related to the original $x[n]$, e.g., "delayed by 4" or "flipped and then advanced by 3."

■ 1.4 Properties of Discrete-Time Systems

Discrete-time systems are often characterized in terms of a number of properties such as linearity, time invariance, stability, causality, and invertibility. It is important to understand how to demonstrate when a system does or does not satisfy a given property. MATLAB can be used to construct counter-examples demonstrating that certain properties are not satisfied. In this exercise, you will obtain practice using MATLAB to construct such counter-examples for a variety of systems and properties.

Basic Problems

For these problems, you are told which property a given system does not satisfy, and the input sequence or sequences that demonstrate clearly how the system violates the property. For each system, define MATLAB vectors representing the input(s) and output(s). Then, make plots of these signals, and construct a well reasoned argument explaining how these figures demonstrate that the system fails to satisfy the property in question.

(a). The system $y[n] = \sin((\pi/2)x[n])$ is not linear. Use the signals $x_1[n] = \delta[n]$ and $x_2[n] = 2\delta[n]$ to demonstrate how the system violates linearity.

(b). The system $y[n] = x[n] + x[n+1]$ is not causal. Use the signal $x[n] = u[n]$ to demonstrate this. Define the MATLAB vectors x and y to represent the input on the interval $-5 \leq n \leq 9$, and the output on the interval $-6 \leq n \leq 9$, respectively.

Intermediate Problems

For these problems, you will be given a system and a property that the system does not satisfy, but must discover for yourself an input or pair of input signals to base your argument upon. Again, create MATLAB vectors to represent the inputs and outputs of the system and generate appropriate plots with these vectors. Use your plots to make a clear and concise argument about why the system does not satisfy the specified property.

(c). The system $y[n] = \log(x[n])$ is not stable.

(d). The system given in Part (a) is not invertible.

Advanced Problems

For each of the following systems, state whether or not the system is linear, time-invariant, causal, stable, and invertible. For each property you claim the system does not possess,

construct a counter-argument using MATLAB to demonstrate how the system violates the property in question.

(e). $y[n] = x^3[n]$

(f). $y[n] = n\,x[n]$

(g). $y[n] = x[2n]$.

■ 1.5 Implementing a First-Order Difference Equation

Discrete-time systems are often implemented with linear constant-coefficient difference equations. Two very simple difference equations are the first-order moving average

$$y[n] = x[n] + b\,x[n-1]\,, \tag{1.5}$$

and the first-order autoregression

$$y[n] = a\,y[n-1] + x[n]\,. \tag{1.6}$$

Even these simple systems can be used to model or approximate a number of practical systems. For instance, the first-order autoregression can be used to model a bank account, where $y[n]$ is the balance at time n, $x[n]$ is the deposit or withdrawal at time n, and $a = 1+r$ is the compounding due to interest rate r. In this exercise, you will be asked to write a function which implements the first-order autoregression equation. You will then be asked to test and analyze your function on some example systems.

Advanced Problems

(a). Write a function `y=diffeqn(a,x,yn1)` which computes the output $y[n]$ of the causal system determined by Eq. (1.6). The input vector `x` contains $x[n]$ for $0 \le n \le N-1$ and `yn1` supplies the value of $y[-1]$. The output vector `y` contains $y[n]$ for $0 \le n \le N-1$. The first line of your M-file should read

```
function y = diffeqn(a,x,yn1)
```

Hint: Note that $y[-1]$ is necessary for computing $y[0]$, which is the first step of the autoregression. Use a `for` loop in your M-file to compute $y[n]$ for successively larger values of n, starting with $n = 0$.

(b). Assume that $a = 1$, $y[-1] = 0$, and that we are only interested in the output over the interval $0 \le n \le 30$. Use your function to compute the response due to $x_1[n] = \delta[n]$ and $x_2[n] = u[n]$, the unit impulse and unit step, respectively. Plot each response using `stem`.

(c). Assume again that $a = 1$, but that $y[-1] = -1$. Use your function to compute $y[n]$ over $0 \le n \le 30$ when the inputs are $x_1[n] = u[n]$ and $x_2[n] = 2\,u[n]$. Define the outputs produced by the two signals to be $y_1[n]$ and $y_2[n]$, respectively. Use `stem` to display both outputs. Use `stem` to plot $(2\,y_1[n] - y_2[n])$. Given that Eq. (1.6) is a linear difference equation, why isn't this difference identically zero?

(d). The causal systems described by Eq. (1.6) are BIBO (bounded-input bounded-output) stable whenever $|a| < 1$. A property of these stable systems is that the effect of the initial condition becomes insignificant for sufficiently large n. Assume $a = 1/2$ and that x contains $x[n] = u[n]$ for $0 \leq n \leq 30$. Assuming both $y[-1] = 0$ and $y[-1] = 1/2$, compute the two output signals $y[n]$ for $0 \leq n \leq 30$. Use `stem` to display both responses. How do they differ?