CHAPTER 1

Introduction

During the past several decades the field of digital signal processing (DSP) has grown to be important, both theoretically and technologically. A major reason for its success in industry is the development and use of low-cost software and hardware. New technologies and applications in various fields are now taking advantage of DSP algorithms. This will lead to a greater demand for electrical and computer engineers with background in DSP. Therefore, it is necessary to make DSP an integral part of any electrical engineering curriculum.

Two decades ago an introductory course on DSP was given mainly at the graduate level. It was supplemented by computer exercises on filter design, spectrum estimation, and related topics using mainframe (or mini) computers. However, considerable advances in personal computers and software during the past two decades have made it necessary to introduce a DSP course to undergraduates. Since DSP applications are primarily algorithms that are implemented either on a DSP processor [11] or in software, a fair amount of programming is required. Using interactive software, such as MATLAB, it is now possible to place more emphasis on learning new and difficult concepts than on programming algorithms. Interesting practical examples can be discussed, and useful problems can be explored.

With this philosophy in mind, we have developed this book as a *companion book* (to traditional textbooks like [18, 23]) in which MATLAB is an integral part in the discussion of topics and concepts. We have chosen MATLAB as the programming tool primarily because of its wide availability on computing platforms in many universities across the world. Furthermore, a low-cost student version of MATLAB has been available for several years, placing it among the least expensive software products for educational purposes. We have treated MATLAB as a computational and programming toolbox containing several tools (sort of a super calculator with several keys) that can be used to explore and solve problems and, thereby, enhance the learning process.

This book is written at an introductory level in order to introduce undergraduate students to an exciting and practical field of DSP. We emphasize that this is not a textbook in the traditional sense but a companion book in which more attention is given to problem solving and hands-on experience with MATLAB. Similarly, it is not a tutorial book in MATLAB. We assume that the student is familiar with MATLAB and is currently taking a course in DSP. The book provides basic analytical tools needed to process real-world signals (a.k.a. analog signals) using digital techniques. We deal mostly with discrete-time signals and systems, which are analyzed in both the time and the frequency domains. The analysis and design of processing structures called *filters* and *spectrum analyzers* are among of the most important aspects of DSP and are treated in great detail in this book. Two important topics on finite word-length effects and sampling-rate conversion are also discussed in this book. More advanced topics in modern signal processing like statistical and adaptive signal processing are generally covered in a graduate course. These are not treated in this book, but it is hoped that the experience gained in using this book will allow students to tackle advanced topics with greater ease and understanding. In this chapter we provide a brief overview of both DSP and MATLAB.

1.1 OVERVIEW OF DIGITAL SIGNAL PROCESSING

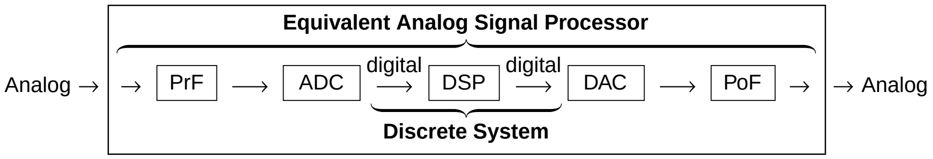
In this modern world we are surrounded by all kinds of signals in various forms. Some of the signals are natural, but most of the signals are manmade. Some signals are necessary (speech), some are pleasant (music), while many are unwanted or unnecessary in a given situation. In an engineering context, signals are carriers of information, both useful and unwanted. Therefore extracting or enhancing the useful information from a mix of conflicting information is the simplest form of signal processing. More generally, signal processing is an operation designed for extracting, enhancing, storing, and transmitting useful information. The distinction between useful and unwanted information is often subjective as well as objective. Hence signal processing tends to be application dependent.

1.1.1 HOW ARE SIGNALS PROCESSED?

The signals that we encounter in practice are mostly analog signals. These signals, which vary continuously in time and amplitude, are processed using electrical networks containing active and passive circuit elements. This approach is known as analog signal processing (ASP)—for example, radio and television receivers.

$$\text{Analog signal: } x_a(t) \longrightarrow \boxed{\text{Analog signal processor}} \longrightarrow y_a(t) \text{ :Analog signal}$$

They can also be processed using digital hardware containing adders, multipliers, and logic elements or using special-purpose microprocessors. However, one needs to convert analog signals into a form suitable for digital hardware. This form of the signal is called a digital signal. It takes one of the finite number of values at specific instances in time, and hence it can be represented by binary numbers, or bits. The processing of digital signals is called DSP; in block diagram form it is represented by



The various block elements are discussed as follows.

PrF: This is a prefilter or an antialiasing filter, which conditions the analog signal to prevent aliasing.

ADC: This is an analog-to-digital converter, which produces a stream of binary numbers from analog signals.

Digital Signal Processor: This is the heart of DSP and can represent a general-purpose computer or a special-purpose processor, or digital hardware, and so on.

DAC: This is the inverse operation to the ADC, called a digital-to-analog converter, which produces a staircase waveform from a sequence of binary numbers, a first step toward producing an analog signal.

PoF: This is a postfilter to smooth out staircase waveform into the desired analog signal.

It appears from the above two approaches to signal processing, analog and digital, that the DSP approach is the more complicated, containing more components than the “simpler looking” ASP. Therefore one might ask, Why process signals digitally? The answer lies in the many advantages offered by DSP.

1.1.2 ADVANTAGES OF DSP OVER ASP

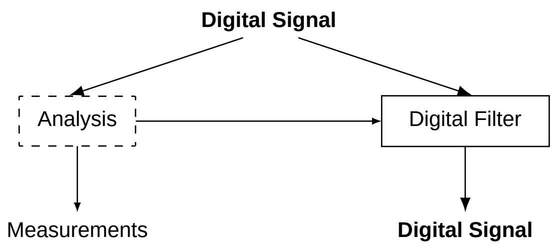
A major drawback of ASP is its limited scope for performing complicated signal-processing applications. This translates into nonflexibility in processing and complexity in system designs. All of these generally lead to expensive products. On the other hand, using a DSP approach, it is possible to convert an inexpensive personal computer into a powerful signal processor. Some important advantages of DSP are these:

1. Systems using the DSP approach can be developed using software running on a general-purpose computer. Therefore DSP is relatively convenient to develop and test, and the software is portable.
2. DSP operations are based solely on additions and multiplications, leading to extremely stable processing capability—for example, stability independent of temperature.
3. DSP operations can easily be modified in real time, often by simple programming changes, or by reloading of registers.
4. DSP has lower cost due to VLSI technology, which reduces costs of memories, gates, microprocessors, and so forth.

The principal disadvantage of DSP is the limited speed of operations limited by the DSP hardware, especially at very high frequencies. Primarily because of its advantages, DSP is now becoming a first choice in many technologies and applications, such as consumer electronics, communications, wireless telephones, and medical imaging.

1.1.3 TWO IMPORTANT CATEGORIES OF DSP

Most DSP operations can be categorized as being either signal *analysis* tasks or signal *filtering* tasks:



Signal analysis This task deals with the measurement of signal properties. It is generally a frequency-domain operation. Some of its applications are

- spectrum (frequency and/or phase) analysis
- speech recognition
- speaker verification
- target detection

Signal filtering This task is characterized by the signal-in signal-out situation. The systems that perform this task are generally called *filters.*

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

$$\texttt{x} * \texttt{y} \Rightarrow \mathbf{xy} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix} = \begin{bmatrix} x_1y_1 & \cdots & x_1y_M \\ \vdots & \ddots & \vdots \\ x_Ny_1 & \cdots & x_Ny_M \end{bmatrix}$$

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

$$\texttt{y} * \texttt{x} \Rightarrow \mathbf{yx} = \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} = x_1y_1 + \cdots + x_My_M$$

- **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

$$\texttt{x.*y} \rightarrow \text{1D array}$$

$$\texttt{X.*Y} \rightarrow \text{2D array}$$

- **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

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

which can be written in matrix form as

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

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

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)

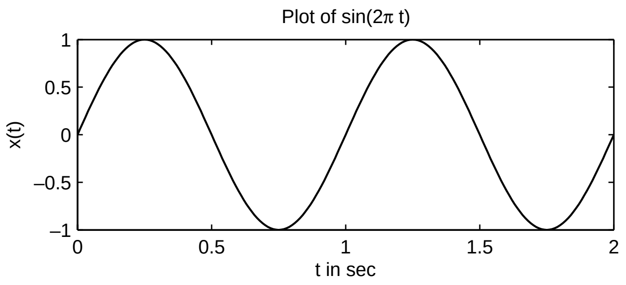


FIGURE 1.1 *Plot of the* $\sin(2\pi t)$ *function*

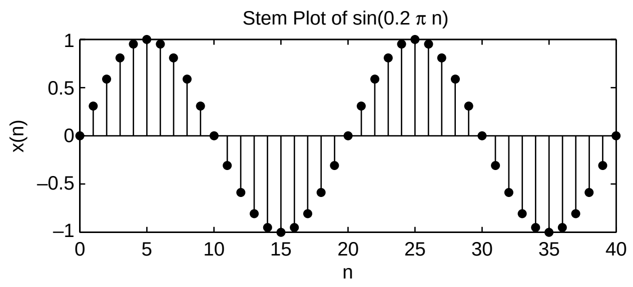


FIGURE 1.2 *Plot of the sin(0.2π n) sequence*

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
. . .
```

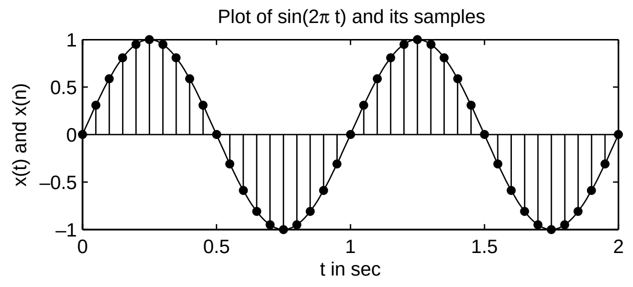


FIGURE 1.3 *Simultaneous plots of $x(t)$ and $x(n)$*

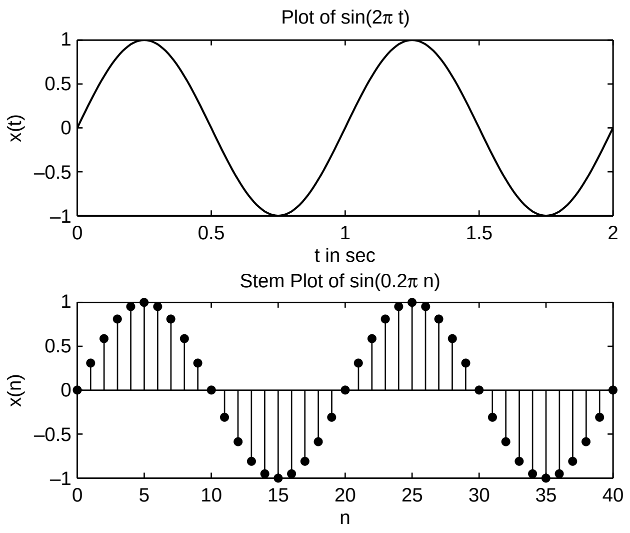


FIGURE 1.4 *Plots of $x(t)$ and $x(n)$ in two rows*

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.)

- consumer electronics (cellular/mobile phones, digital television, digital camera, Internet voice/music/video, interactive entertainment systems, etc) and many more.

These applications and products require many interconnected complex steps, such as collection, processing, transmission, analysis, audio/display of real-world information in near real time. DSP technology has made it possible to incorporate these steps into devices that are innovative, affordable, and of high quality (for example, iPhone from Apple, Inc.). A typical application to music is now considered as a motivation for the study of DSP.

Musical sound processing In the music industry, almost all musical products (songs, albums, etc.) are produced in basically two stages. First, the sound from an individual instrument or performer is recorded in an acoustically inert studio on a single track of a multitrack recording device. Then, stored signals from each track are digitally processed by the sound engineer by adding special effects and combined into a stereo recording, which is then made available either on a CD or as an audio file.

The audio effects are artificially generated using various signal-processing techniques. These effects include echo generation, reverberation (concert hall effect), flanging (in which audio playback is slowed down by placing DJ's thumb on the *flange* of the feed reel), chorus effect (when several musicians play the same instrument with small changes in amplitudes and delays), and phasing (aka phase shifting, in which an audio effect takes advantage of how sound waves interact with each other when they are out of phase). These effects are now generated using digital-signal-processing techniques. We now discuss a few of these sound effects in some detail.

Echo Generation The most basic of all audio effects is that of *time delay*, or echoes. It is used as the building block of more complicated effects such as reverb or flanging. In a listening space such as a room, sound waves arriving at our ears consist of *direct* sound from the source as well as *reflected* off the walls, arriving with different amounts of attenuation and delays.

Echoes are delayed signals, and as such are generated using delay units. For example, the combination of the direct sound represented by discrete signal $y[n]$ and a single echo appearing D samples later (which is related to delay in seconds) can be generated by the equation of the form (called a difference equation)

$$x[n] = y[n] + \alpha y[n-D], \qquad |\alpha| < 1 \tag{1.2}$$

where $x[n]$ is the resulting signal and α models attenuation of the direct sound. Difference equations are implemented in MATLAB using the `filter` function. Available in MATLAB is a short snippet of Handel's hallelujah chorus, which is a digital sound about 9 seconds long, sampled at 8192 sam/sec. To experience the sound with echo in (1.2), execute the following fragment at the command window. The echo is delayed by $D = 4196$ samples, which amount to 0.5 sec of delay.

```
load handel; % the signal is in y and sampling freq in Fs
sound(y,Fs); pause(10); % Play the original sound
alpha = 0.9; D = 4196; % Echo parameters
b = [1,zeros(1,D),alpha]; % Filter parameters
x = filter(b,1,y); % Generate sound plus its echo
sound(x,Fs); % Play sound with echo
```

You should be able to hear the distinct echo of the chorus in about a half second.

Echo Removal After executing this simulation, you may experience that the echo is an objectionable interference while listening. Again DSP can be used effectively to reduce (almost eliminate) echoes. Such an echo-removal system is given by the difference equation

$$w[n] + \alpha w[n - D] = x[n] \tag{1.3}$$

where $x[n]$ is the echo-corrupted sound signal and $w[n]$ is the output sound signal, which has the echo (hopefully) removed. Note again that this system is very simple to implement in software or hardware. Now try the following MATLAB script on the signal $x[n]$.

```
w = filter(1,b,x);
sound(w,Fs)
```

The echo should no longer be audible.

Digital Reverberation Multiple close-spaced echoes eventually lead to reverberation, which can be created digitally using a somewhat more involved difference equation

$$x[n] = \sum_{k=0}^{N-1} \alpha^k y[n - kD] \tag{1.4}$$

which generates multiple echoes spaced D samples apart with exponentially decaying amplitudes. Another natural sounding reverberation is given by

$$x[n] = \alpha y[n] + y[n-D] + \alpha x[n-D], \quad |\alpha| < 1 \tag{1.5}$$

which simulates a higher echo density.

These simple applications are examples of DSP. Using techniques, concepts, and MATLAB functions learned in this book you should be able to simulate these and other interesting sound effects.

1.4 BRIEF OVERVIEW OF THE BOOK

The first part of this book, which comprises Chapters 2 through 5, deals with the signal-analysis aspect of DSP. Chapter 2 begins with basic descriptions of discrete-time signals and systems. These signals and systems are analyzed in the frequency domain in Chapter 3. A generalization of the frequency-domain description, called the z-transform, is introduced in Chapter 4. The practical algorithms for computing the Fourier transform are discussed in Chapter 5 in the form of the discrete Fourier transform and the fast Fourier transform.

Chapters 6 through 8 constitute the second part of this book, which is devoted to the signal-filtering aspect of DSP. Chapter 6 describes various implementations and structures of digital filters. It also introduces finite-precision number representation, filter coefficient quantization, and its effect on filter performance. Chapter 7 introduces design techniques and algorithms for designing one type of digital filter called *finite-duration impulse response (FIR) filters*, and Chapter 8 provides a similar treatment for another type of filter called *infinite-duration impulse response (IIR) filters*. In both chapters only the simpler but practically useful techniques of filter design are discussed. More advanced techniques are not covered.

Finally, the last part, which consists of the remaining four chapters, provides important topics and applications in DSP. Chapter 9 deals with the useful topic of the sampling-rate conversion and applies FIR filter designs from Chapter 7 to design practical sample-rate converters. Chapter 10 extends the treatment of finite-precision numerical representation to signal quantization and the effect of finite-precision arithmetic on filter performance. The last two chapters provide some practical applications in the form of projects that can be done using material presented in the first 10 chapters. In Chapter 11, concepts in adaptive filtering are introduced, and simple projects in system identification, interference suppression, adaptive line enhancement, and so forth are discussed. In Chapter 12 a brief introduction to digital communications is presented with projects involving such topics as PCM, DPCM, and LPC being outlined.

In all these chapters, the central theme is the generous use and adequate demonstration of MATLAB, which can be used as an effective teaching as well as learning tool. Most of the existing MATLAB functions for DSP are described in detail, and their correct use is demonstrated in many examples. Furthermore, many new MATLAB functions are developed to provide insights into the working of many algorithms. The authors believe that this hand-holding approach enables students to dispel fears about DSP and provides an enriching learning experience.

CHAPTER 2

Discrete-time Signals and Systems

We begin with the concepts of signals and systems in discrete time. A number of important types of signals and their operations are introduced. Linear and shift-invariant systems are discussed mostly because they are easier to analyze and implement. The convolution and the difference equation representations are given special attention because of their importance in digital signal processing and in MATLAB. The emphasis in this chapter is on the representations and implementation of signals and systems using MATLAB.

2.1 DISCRETE-TIME SIGNALS

Signals are broadly classified into analog and discrete signals. An analog signal will be denoted by $x_a(t)$, in which the variable t can represent any physical quantity, but we will assume that it represents time in seconds. A discrete signal will be denoted by $x(n)$, in which the variable n is integer-valued and represents discrete instances in time. Therefore it is also called a discrete-time signal, which is a *number sequence* and will be denoted by one of the following notations:

$$x(n) = \{x(n)\} = \{\ldots, x(-1), \underset{\uparrow}{x(0)}, x(1), \ldots\}$$

where the *up-arrow* indicates the sample at $n = 0$.

In MATLAB we can represent a *finite-duration* sequence by a *row vector* of appropriate values. However, such a vector does not have any information about sample position n. Therefore a correct representation of $x(n)$ would require two vectors, one each for x and n. For example, a sequence $x(n) = \{2, 1, -1, \underset{\uparrow}{0}, 1, 4, 3, 7\}$ can be represented in MATLAB by

```
>> n=[-3,-2,-1,0,1,2,3,4];  x=[2,1,-1,0,1,4,3,7];
```

Generally, we will use the x-vector representation alone when the sample position information is not required or when such information is trivial (e.g. when the sequence begins at $n = 0$). An arbitrary *infinite-duration* sequence cannot be represented in MATLAB due to the finite memory limitations.

2.1.1 TYPES OF SEQUENCES

We use several elementary sequences in digital signal processing for analysis purposes. Their definitions and MATLAB representations follow.

1. **Unit sample sequence**:

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} = \left\{ \dots, 0, 0, \underset{\uparrow}{1}, 0, 0, \dots \right\}$$

In MATLAB the function `zeros(1,N)` generates a row vector of N zeros, which can be used to implement $\delta(n)$ over a finite interval. However, the logical relation `n==0` is an elegant way of implementing $\delta(n)$. For example, to implement

$$\delta(n - n_0) = \begin{cases} 1, & n = n_0 \\ 0, & n \neq n_0 \end{cases}$$

over the $n_1 \leq n_0 \leq n_2$ interval, we will use the following MATLAB function.

```
function [x,n] = impseq(n0,n1,n2)
% Generates x(n) = delta(n-n0); n1 <= n <= n2
% ----------------------------------------------
% [x,n] = impseq(n0,n1,n2)
%
n = [n1:n2]; x = [(n-n0) == 0];
```

2. **Unit step sequence**:

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} = \{ \dots, 0, 0, \underset{\uparrow}{1}, 1, 1, \dots \}$$

In MATLAB the function `ones(1,N)` generates a row vector of N ones. It can be used to generate $u(n)$ over a finite interval. Once again an elegant approach is to use the logical relation `n>=0`. To implement

$$u(n-n_0) = \begin{cases} 1, & n \geq n_0 \\ 0, & n < n_0 \end{cases}$$

over the $n_1 \leq n_0 \leq n_2$ interval, we will use the following MATLAB function.

```
function [x,n] = stepseq(n0,n1,n2)
% Generates x(n) = u(n-n0); n1 <= n <= n2
% ------------------------------------------
% [x,n] = stepseq(n0,n1,n2)
%
n = [n1:n2]; x = [(n-n0) >= 0];
```

3. **Real-valued exponential sequence**:

$$x(n) = a^n, \forall n; \quad a \in \mathbb{R}$$

In MATLAB an array operator "`.^`" is required to implement a real exponential sequence. For example, to generate $x(n) = (0.9)^n$, $0 \leq n \leq 10$, we will need the following MATLAB script:

```
>> n = [0:10]; x = (0.9).^n;
```

4. **Complex-valued exponential sequence**:

$$x(n) = e^{(\sigma + j\omega_0)n}, \forall n$$

where σ produces an attenuation (if <0) or amplification (if >0) and ω_0 is the frequency in radians. A MATLAB function `exp` is used to generate exponential sequences. For example, to generate $x(n) = \exp[(2+j3)n], 0 \leq n \leq 10$, we will need the following MATLAB script:

```
>> n = [0:10]; x = exp((2+3j)*n);
```

5. **Sinusoidal sequence**:

$$x(n) = A\cos(\omega_0 n + \theta_0), \forall n$$

where A is an amplitude and θ_0 is the phase in radians. A MATLAB function `cos` (or `sin`) is used to generate sinusoidal sequences.

For example, to generate $x(n) = 3\cos(0.1\pi n + \pi/3) + 2\sin(0.5\pi n)$, $0 \le n \le 10$, we will need the following MATLAB script:

```
>> n = [0:10]; x = 3*cos(0.1*pi*n+pi/3) + 2*sin(0.5*pi*n);
```

6. **Random sequences**: Many practical sequences cannot be described by mathematical expressions like those above. These sequences are called random (or stochastic) sequences and are characterized by parameters of the associated probability density functions. In MATLAB two types of (pseudo-) random sequences are available. The `rand(1,N)` generates a length N random sequence whose elements are uniformly distributed between $[0, 1]$. The `randn(1,N)` generates a length N Gaussian random sequence with mean 0 and variance 1. Other random sequences can be generated using transformations of the above functions.
7. **Periodic sequence**: A sequence $x(n)$ is periodic if $x(n) = x(n+N)$, $\forall n$. The smallest integer N that satisfies this relation is called the *fundamental* period. We will use $\tilde{x}(n)$ to denote a periodic sequence. To generate P periods of $\tilde{x}(n)$ from one period $\{x(n), \quad 0 \le n \le N-1\}$, we can copy $x(n)$ P times:

```
>> xtilde = [x,x,...,x];
```

But an elegant approach is to use MATLAB's powerful indexing capabilities. First we generate a matrix containing P rows of $x(n)$ values. Then we can concatenate P rows into a long row vector using the construct `(:)`. However, this construct works only on columns. Hence we will have to use the matrix transposition operator `'` to provide the same effect on rows.

```
>> xtilde = x' * ones(1,P);   % P columns of x; x is a row vector
>> xtilde = xtilde(:);        % long column vector
>> xtilde = xtilde';          % long row vector
```

Note that the last two lines can be combined into one for compact coding. This is shown in Example 2.1.

2.1.2 OPERATIONS ON SEQUENCES

Here we briefly describe basic sequence operations and their MATLAB equivalents.

1. **Signal addition**: This is a sample-by-sample addition given by

$$\{x_1(n)\} + \{x_2(n)\} = \{x_1(n) + x_2(n)\}$$

It is implemented in MATLAB by the arithmetic operator "+". However, the lengths of $x_1(n)$ and $x_2(n)$ must be the same. If sequences are of unequal lengths, or if the sample positions are different for equal-length sequences, then we cannot directly use the operator +. We have to first augment $x_1(n)$ and $x_2(n)$ so that they have the same position vector n (and hence the same length). This requires careful attention to MATLAB's indexing operations. In particular, logical operation of intersection "&", relational operations like "<=" and "==", and the find function are required to make $x_1(n)$ and $x_2(n)$ of equal length. The following function, called the sigadd function, demonstrates these operations.

```
function [y,n] = sigadd(x1,n1,x2,n2)
% implements y(n) = x1(n)+x2(n)
% -----------------------------
% [y,n] = sigadd(x1,n1,x2,n2)
%   y = sum sequence over n, which includes n1 and n2
%  x1 = first sequence over n1
%  x2 = second sequence over n2 (n2 can be different from n1)
%
n = min(min(n1),min(n2)):max(max(n1),max(n2));  % duration of y(n)
y1 = zeros(1,length(n)); y2 = y1;               % initialization
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;      % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;      % x2 with duration of y
y = y1+y2;                                      % sequence addition
```

Its use is illustrated in Example 2.2.

2. **Signal multiplication**: This is a sample-by-sample (or "dot") multiplication) given by

$$\{x_1(n)\} \cdot \{x_2(n)\} = \{x_1(n)x_2(n)\}$$

It is implemented in MATLAB by the array operator .*. Once again, the similar restrictions apply for the .* operator as for the + operator. Therefore we have developed the sigmult function, which is similar to the sigadd function.

```
function [y,n] = sigmult(x1,n1,x2,n2)
% implements y(n) = x1(n)*x2(n)
% -----------------------------
% [y,n] = sigmult(x1,n1,x2,n2)
%   y = product sequence over n, which includes n1 and n2
%  x1 = first sequence over n1
%  x2 = second sequence over n2 (n2 can be different from n1)
%
```

```
n = min(min(n1),min(n2)):max(max(n1),max(n2));  % duration of y(n)
y1 = zeros(1,length(n)); y2 = y1;               %
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;      % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;      % x2 with duration of y
y = y1 .* y2;                                   % sequence multiplication
```

Its use is also given in Example 2.2.

3. **Scaling**: In this operation each sample is multiplied by a scalar α.

$$\alpha\{x(n)\} = \{\alpha x(n)\}$$

An arithmetic operator (`*`) is used to implement the scaling operation in MATLAB.

4. **Shifting**: In this operation, each sample of $x(n)$ is shifted by an amount k to obtain a shifted sequence $y(n)$.

$$y(n) = \{x(n-k)\}$$

If we let $m = n-k$, then $n = m+k$ and the above operation is given by

$$y(m+k) = \{x(m)\}$$

Hence this operation has no effect on the vector `x`, but the vector `n` is changed by adding k to each element. This is shown in the function `sigshift`.

```
function [y,n] = sigshift(x,m,k)
% implements y(n) = x(n-k)
% -------------------------
% [y,n] = sigshift(x,m,k)
%
n = m+k; y = x;
```

Its use is given in Example 2.2.

5. **Folding**: In this operation each sample of $x(n)$ is flipped around $n = 0$ to obtain a folded sequence $y(n)$.

$$y(n) = \{x(-n)\}$$

In MATLAB this operation is implemented by `fliplr(x)`function for sample values and by `-fliplr(n)` function for sample positions as shown in the `sigfold` function.

```
function [y,n] = sigfold(x,n)
% implements y(n) = x(-n)
% -----------------------
% [y,n] = sigfold(x,n)
%
y = fliplr(x); n = -fliplr(n);
```

6. **Sample summation**: This operation differs from signal addition operation. It adds all sample values of $x(n)$ between n_1 and n_2.

$$\sum_{n=n_1}^{n_2} x(n) = x(n_1) + \cdots + x(n_2)$$

   It is implemented by the `sum(x(n1:n2))` function.
7. **Sample products**: This operation also differs from signal multiplication operation. It multiplies all sample values of $x(n)$ between n_1 and n_2.

$$\prod_{n_1}^{n_2} x(n) = x(n_1) \times \cdots \times x(n_2)$$

   It is implemented by the `prod(x(n1:n2))` function.
8. **Signal energy**: The energy of a sequence $x(n)$ is given by

$$\mathcal{E}_x = \sum_{-\infty}^{\infty} x(n)x^*(n) = \sum_{-\infty}^{\infty} |x(n)|^2$$

   where superscript * denotes the operation of complex conjugation.[1] The energy of a finite-duration sequence $x(n)$ can be computed in MATLAB using

```
>> Ex = sum(x .* conj(x)); % one approach
>> Ex = sum(abs(x) .^ 2); % another approach
```

9. **Signal power**: The average power of a periodic sequence $\tilde{x}(n)$ with fundamental period N is given by

$$\mathcal{P}_x = \frac{1}{N} \sum_{0}^{N-1} |\tilde{x}(n)|^2$$

□ **EXAMPLE 2.1** Generate and plot each of the following sequences over the indicated interval.

a. $x(n) = 2\delta(n+2) - \delta(n-4), \quad -5 \le n \le 5.$
b. $x(n) = n[u(n) - u(n-10)] + 10e^{-0.3(n-10)}[u(n-10) - u(n-20)], \; 0 \le n \le 20.$
c. $x(n) = \cos(0.04\pi n) + 0.2w(n), \; 0 \le n \le 50$, where $w(n)$ is a Gaussian random sequence with zero mean and unit variance.
d. $\tilde{x}(n) = \{..., 5, 4, 3, 2, 1, \underset{\uparrow}{5}, 4, 3, 2, 1, 5, 4, 3, 2, 1, ...\}; \; -10 \le n \le 9.$

[1]The symbol * denotes many operations in digital signal processing. Its font (roman or computer) and its position (normal or superscript) will distinguish each operation.

Solution

a. $x(n) = 2\delta(n+2) - \delta(n-4), \quad -5 \le n \le 5.$

```
>> n = [-5:5];
>> x = 2*impseq(-2,-5,5) - impseq(4,-5,5);
>> stem(n,x); title('Sequence in Problem 2.1a')
>> xlabel('n'); ylabel('x(n)');
```

The plot of the sequence is shown in Figure 2.1a.

b. $x(n) = n\,[u(n) - u(n-10)] + 10e^{-0.3(n-10)}\,[u(n-10) - u(n-20)],\ 0 \le n \le 20.$

```
>> n = [0:20]; x1 = n.*(stepseq(0,0,20)-stepseq(10,0,20));
>> x2 = 10*exp(-0.3*(n-10)).*(stepseq(10,0,20)-stepseq(20,0,20));
>> x = x1+x2;
>> subplot(2,2,3); stem(n,x); title('Sequence in Problem 2.1b')
>> xlabel('n'); ylabel('x(n)');
```

The plot of the sequence is shown in Figure 2.1b.

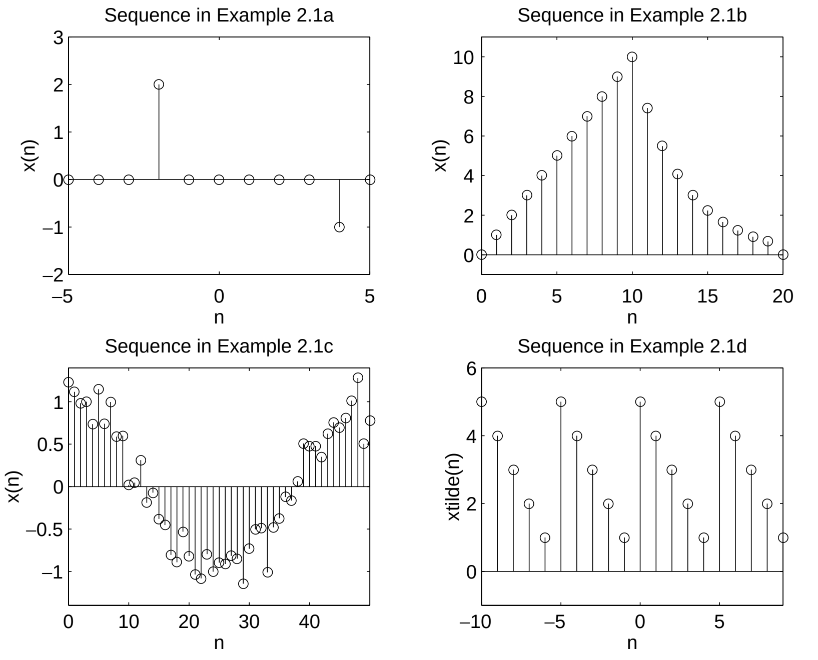


FIGURE 2.1 *Sequences in Example 2.1*

c. $x(n) = \cos(0.04\pi n) + 0.2w(n), \quad 0 \le n \le 50.$

```
>> n = [0:50]; x = cos(0.04*pi*n)+0.2*randn(size(n));
>> subplot(2,2,2); stem(n,x); title('Sequence in Problem 2.1c')
>> xlabel('n'); ylabel('x(n)');
```

The plot of the sequence is shown in Figure 2.1c.

d. $\tilde{x}(n) = \{..., 5, 4, 3, 2, 1, \underset{\uparrow}{5}, 4, 3, 2, 1, 5, 4, 3, 2, 1, ...\}; \ -10 \le n \le 9.$

Note that over the given interval, the sequence $\tilde{x}(n)$ has four periods.

```
>> n = [-10:9]; x = [5,4,3,2,1];
>> xtilde = x' * ones(1,4);   xtilde = (xtilde(:))';
>> subplot(2,2,4); stem(n,xtilde); title('Sequence in Problem 2.1d')
>> xlabel('n'); ylabel('xtilde(n)');
```

The plot of the sequence is shown in Figure 2.1d. □

□ **EXAMPLE 2.2** Let $x(n) = \{1, 2, \underset{\uparrow}{3}, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1\}$. Determine and plot the following sequences.

a. $x_1(n) = 2x(n-5) - 3x(n+4)$
b. $x_2(n) = x(3-n) + x(n)\,x(n-2)$

Solution The sequence $x(n)$ is nonzero over $-2 \le n \le 10$. Hence

```
>> n = -2:10; x = [1:7,6:-1:1];
```

will generate $x(n)$.

a. $x_1(n) = 2x(n-5) - 3x(n+4)$.
The first part is obtained by shifting $x(n)$ by 5 and the second part by shifting $x(n)$ by -4. This shifting and the addition can be easily done using the `sigshift` and the `sigadd` functions.

```
>> [x11,n11] = sigshift(x,n,5); [x12,n12] = sigshift(x,n,-4);
>> [x1,n1] = sigadd(2*x11,n11,-3*x12,n12);
>> subplot(2,1,1); stem(n1,x1); title('Sequence in Example 2.2a')
>> xlabel('n'); ylabel('x1(n)');
```

The plot of $x_1(n)$ is shown in Figure 2.2a.

b. $x_2(n) = x(3-n) + x(n)\,x(n-2)$.
The first term can be written as $x(-(n-3))$. Hence it is obtained by first folding $x(n)$ and then shifting the result by 3. The second part is a multiplication of $x(n)$ and $x(n-2)$, both of which have the same length but different

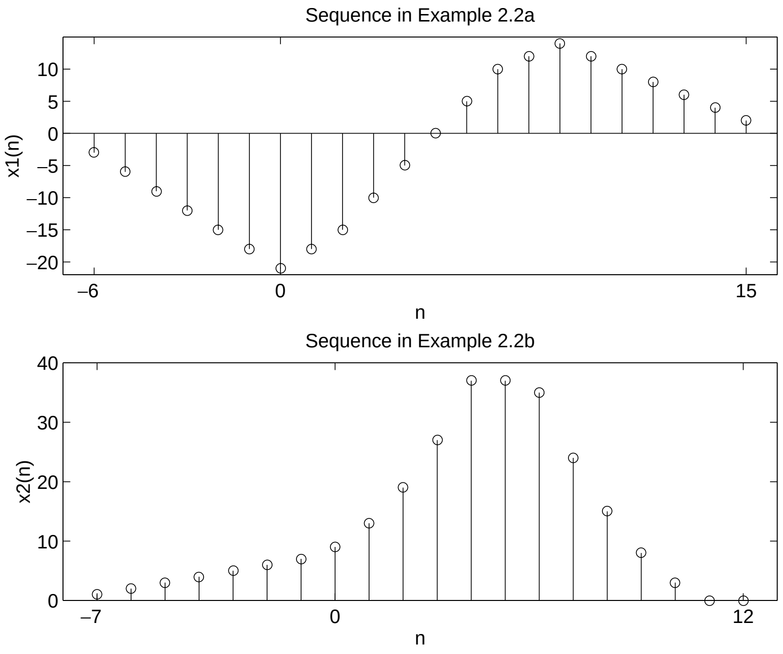


FIGURE 2.2 *Sequences in Example 2.2*

support (or sample positions). These operations can be easily done using the `sigfold` and the `sigmult` functions.

```
>> [x21,n21] = sigfold(x,n); [x21,n21] = sigshift(x21,n21,3);
>> [x22,n22] = sigshift(x,n,2); [x22,n22] = sigmult(x,n,x22,n22);
>> [x2,n2] = sigadd(x21,n21,x22,n22);
>> subplot(2,1,2); stem(n2,x2); title('Sequence in Example 2.2b')
>> xlabel('n'); ylabel('x2(n)');
```

The plot of $x_2(n)$ is shown in Figure 2.2b. □

Example 2.2 shows that the four `sig*` functions developed in this section provide a convenient approach for sequence manipulations.

□ **EXAMPLE 2.3** Generate the complex-valued signal

$$x(n) = e^{(-0.1+j0.3)n}, \quad -10 \le n \le 10$$

and plot its magnitude, phase, the real part, and the imaginary part in four separate subplots.

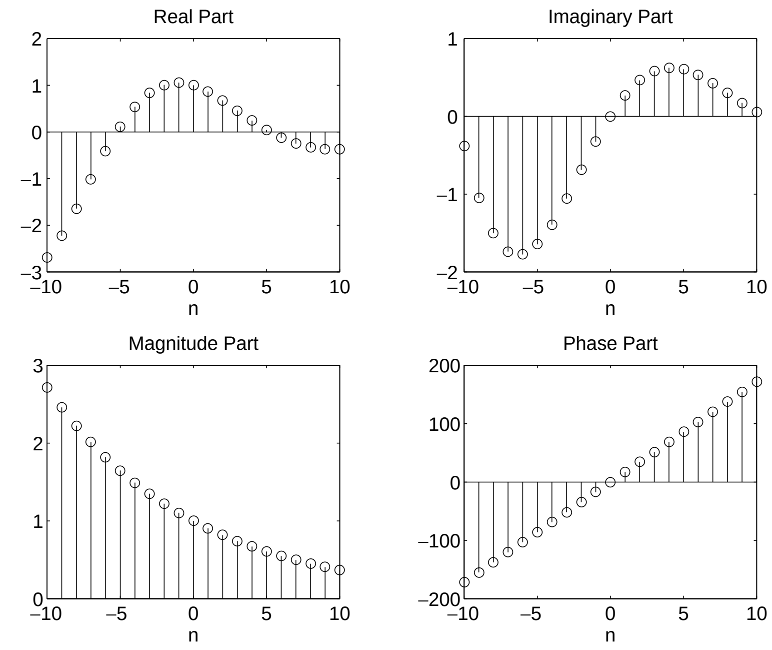


FIGURE 2.3 *Complex-valued sequence plots in Example 2.3*

Solution MATLAB script:

```
>> n = [-10:1:10]; alpha = -0.1+0.3j;
>> x = exp(alpha*n);
>> subplot(2,2,1); stem(n,real(x));title('real part');xlabel('n')
>> subplot(2,2,2); stem(n,imag(x));title('imaginary part');xlabel('n')
>> subplot(2,2,3); stem(n,abs(x));title('magnitude part');xlabel('n')
>> subplot(2,2,4); stem(n,(180/pi)*angle(x));title('phase part');xlabel('n')
```

The plot of the sequence is shown in Figure 2.3. □

2.1.3 DISCRETE-TIME SINUSOIDS

In the last section we introduced the discrete-time sinusoidal sequence $x(n) = A\cos(\omega_0 n + \theta_0)$, for all n as one of the basic signals. This signal is very important in signal theory as a basis for Fourier transform and in system theory as a basis for steady-state analysis. It can be conveniently related to the continuous-time sinusoid $x_a(t) = A\cos(\Omega_0 t + \theta_0)$ using an operation called *sampling* (Chapter 3), in which continuous-time sinusoidal values at equally spaced points $t = nT_s$ are assigned to $x(n)$.

The quantity T_s is called the sampling interval, and $\Omega_0 = \omega_0/T_s$ is called the analog frequency, measured in radians per second.

The fact that n is a discrete variable, whereas t is a continuous variable, leads to some important differences between discrete-time and continuous-time sinusoidal signals.

Periodicity in time From our definition of periodicity, the sinusoidal sequence is periodic if

$$x[n+N] = A\cos(\omega_0 n + \omega_0 N + \theta) = A\cos(\omega_0 n + \theta_0) = x[n] \tag{2.1}$$

This is possible if and only if $\omega_0 N = 2\pi k$, where k is an integer. This leads to the following important result (see Problem P2.5):

> The sequence $x(n) = A\cos(\omega_0 n + \theta_0)$ is periodic if and only if $f_0 \overset{\triangle}{=} \omega_0/2\pi = k/N$, that is, f_0 is a rational number. If k and N are a pair of prime numbers, then N is the fundamental period of $x(n)$ and k represents an integer number of periods kT_s of the corresponding continuous-time sinusoid.

Periodicity in frequency From the definition of the discrete-time sinusoid, we can easily see that

$$\begin{aligned} A\cos[(\omega_0 + k2\pi)n + \theta_0] &= A\cos(\omega_0 n + kn2\pi + \theta_0) \\ &= A\cos(\omega_0 n + \theta_0) \end{aligned}$$

since $(kn)2\pi$ is always an integer multiple of 2π. Therefore, we have the following property:

> The sequence $x(n) = A\cos(\omega_0 n + \theta)$ is periodic in ω_0 with fundamental period 2π and periodic in f_0 with fundamental period one.

This property has a number of very important implications:

1. Sinusoidal sequences with radian frequencies separated by integer multiples of 2π are identical.
2. All distinct sinusoidal sequences have frequencies within an interval of 2π radians. We shall use the so-called *fundamental* frequency ranges

$$-\pi < \omega \le \pi \quad \text{or} \quad 0 \le \omega < 2\pi \tag{2.2}$$

   Therefore, if $0 \le \omega_0 < 2\pi$, the frequencies ω_0 and $\omega_0 + m2\pi$ are indistinguishable from the observation of the corresponding sequences.
3. Since $A\cos[\omega_0(n + n_0) + \theta] = A\cos[\omega_0 n + (\omega_0 n_0 + \theta)]$, a time shift is equivalent to a phase change.
4. The rate of oscillation of a discrete-time sinusoid increases as ω_0 increases from $\omega_0 = 0$ to $\omega_0 = \pi$. However, as ω_0 increases from $\omega_0 = \pi$ to $\omega_0 = 2\pi$, the oscillations become slower. Therefore, low frequencies (slow oscillations) are at the vicinity of $\omega_0 = k2\pi$, and high frequencies (rapid oscillations) are at the vicinity of $\omega_0 = \pi + k2\pi$.

2.1.4 SOME USEFUL RESULTS

There are several important results in discrete-time signal theory. We will discuss some that are useful in digital signal processing.

Unit sample synthesis Any arbitrary sequence $x(n)$ can be synthesized as a weighted sum of delayed and scaled unit sample sequences, such as

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)\delta(n-k) \tag{2.3}$$

We will use this result in the next section.

Even and odd synthesis A real-valued sequence $x_e(n)$ is called even (symmetric) if

$$x_e(-n) = x_e(n)$$

Similarly, a real-valued sequence $x_o(n)$ is called odd (antisymmetric) if

$$x_o(-n) = -x_o(n)$$

Then any arbitrary real-valued sequence $x(n)$ can be decomposed into its even and odd components

$$x(n) = x_e(n) + x_o(n) \tag{2.4}$$

where the even and odd parts are given by

$$x_e(n) = \frac{1}{2}\left[x(n) + x(-n)\right] \quad \text{and} \quad x_o(n) = \frac{1}{2}\left[x(n) - x(-n)\right] \tag{2.5}$$

respectively. We will use this decomposition in studying properties of the Fourier transform. Therefore it is a good exercise to develop a simple MATLAB function to decompose a given sequence into its even and odd components. Using MATLAB operations discussed so far, we can obtain the following `evenodd` function.

```
function [xe, xo, m] = evenodd(x,n)
% Real signal decomposition into even and odd parts
% -------------------------------------------------
% [xe, xo, m] = evenodd(x,n)
%
if any(imag(x) ~= 0)
      error('x is not a real sequence')
end
m = -fliplr(n);
m1 = min([m,n]); m2 = max([m,n]); m = m1:m2;
nm = n(1)-m(1); n1 = 1:length(n);
x1 = zeros(1,length(m));   x1(n1+nm) = x; x = x1;
xe = 0.5*(x + fliplr(x));  xo = 0.5*(x - fliplr(x));
```

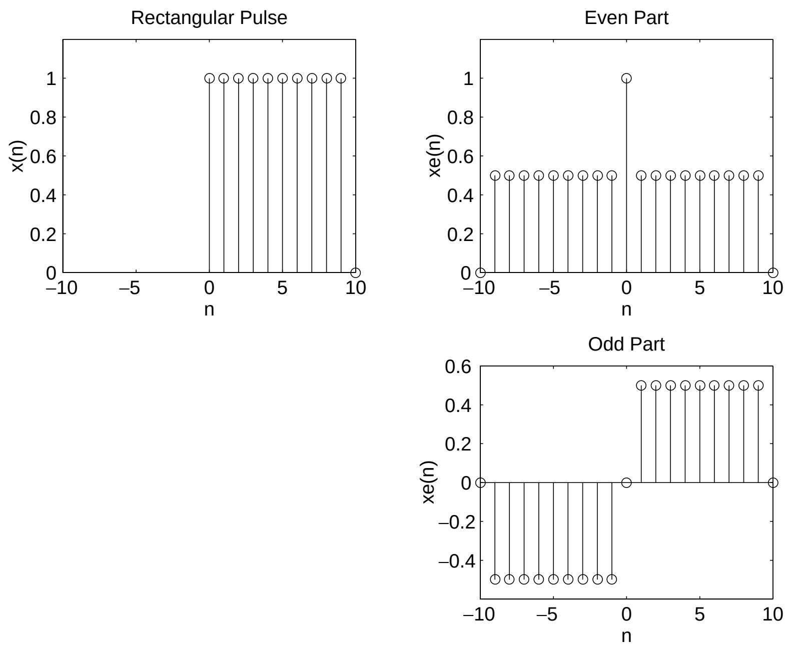


FIGURE 2.4 *Even-odd decomposition in Example 2.4*

The sequence and its support are supplied in `x` and `n` arrays, respectively. It first checks if the given sequence is real and determines the support of the even and odd components in `m` array. It then implements (2.5) with special attention to the MATLAB indexing operation. The resulting components are stored in `xe` and `xo` arrays.

☐ **EXAMPLE 2.4** Let $x(n) = u(n) - u(n-10)$. Decompose $x(n)$ into even and odd components.

Solution The sequence $x(n)$, which is nonzero over $0 \le n \le 9$, is called a *rectangular pulse*. We will use MATLAB to determine and plot its even and odd parts.

```
>> n = [0:10]; x = stepseq(0,0,10)-stepseq(10,0,10);
>> [xe,xo,m] = evenodd(x,n);
>> subplot(2,2,1); stem(n,x); title('Rectangular pulse')
>> xlabel('n'); ylabel('x(n)'); axis([-10,10,0,1.2])
>> subplot(2,2,2); stem(m,xe); title('Even Part')
>> xlabel('n'); ylabel('xe(n)'); axis([-10,10,0,1.2])
>> subplot(2,2,4); stem(m,xo); title('Odd Part')
>> xlabel('n'); ylabel('xe(n)'); axis([-10,10,-0.6,0.6])
```

The plots shown in Figure 2.4 clearly demonstrate the decomposition. ☐

A similar decomposition for complex-valued sequences is explored in Problem P2.5.

The geometric series A one-sided exponential sequence of the form $\{\alpha^n, \quad n \geq 0\}$, where α is an arbitrary constant, is called a geometric series. In digital signal processing, the convergence and expression for the sum of this series are used in many applications. The series converges for $|\alpha| < 1$, while the sum of its components converges to

$$\sum_{n=0}^{\infty} \alpha^n \longrightarrow \frac{1}{1-\alpha}, \quad \text{for } |\alpha| < 1 \tag{2.6}$$

We will also need an expression for the sum of any finite number of terms of the series given by

$$\sum_{n=0}^{N-1} \alpha^n = \frac{1-\alpha^N}{1-\alpha}, \forall \alpha \tag{2.7}$$

These two results will be used throughout this book.

Correlations of sequences Correlation is an operation used in many applications in digital signal processing. It is a measure of the degree to which two sequences are similar. Given two real-valued sequences $x(n)$ and $y(n)$ of finite energy, the *crosscorrelation* of $x(n)$ and $y(n)$ is a sequence $r_{xy}(\ell)$ defined as

$$r_{x,y}(\ell) = \sum_{n=-\infty}^{\infty} x(n)y(n-\ell) \tag{2.8}$$

The index ℓ is called the shift or lag parameter. The special case of (2.8) when $y(n) = x(n)$ is called *autocorrelation* and is defined by

$$r_{xx}(\ell) = \sum_{n=-\infty}^{\infty} x(n)x(n-\ell) \tag{2.9}$$

It provides a measure of self-similarity between different alignments of the sequence. MATLAB functions to compute auto- and crosscorrelations are discussed later in the chapter.

2.2 DISCRETE SYSTEMS

Mathematically, a discrete-time system (or *discrete system* for short) is described as an operator $T[\cdot]$ that takes a sequence $x(n)$ (called *excitation*) and transforms it into another sequence $y(n)$ (called *response*). That is,

$$y(n) = T[x(n)]$$

In DSP we will say that the system processes an *input* signal into an *output* signal. Discrete systems are broadly classified into *linear* and *nonlinear* systems. We will deal mostly with linear systems.

2.2.1 LINEAR SYSTEMS

A discrete system $T[\cdot]$ is a linear operator $L[\cdot]$ if and only if $L[\cdot]$ satisfies the principle of superposition, namely,

$$L[a_1x_1(n) + a_2x_2(n)] = a_1L[x_1(n)] + a_2L[x_2(n)], \forall a_1, a_2, x_1(n), x_2(n) \tag{2.10}$$

Using (2.3) and (2.10), the output $y(n)$ of a linear system to an arbitrary input $x(n)$ is given by

$$y(n) = L[x(n)] = L\left[\sum_{n=-\infty}^{\infty} x(k)\,\delta(n-k)\right] = \sum_{n=-\infty}^{\infty} x(k)L[\delta(n-k)]$$

The response $L[\delta(n-k)]$ can be interpreted as the response of a linear system at time n due to a unit sample (a well-known sequence) at time k. It is called an *impulse response* and is denoted by $h(n,k)$. The output then is given by the *superposition summation*

$$y(n) = \sum_{n=-\infty}^{\infty} x(k)h(n,k) \tag{2.11}$$

The computation of (2.11) requires the *time-varying* impulse response $h(n,k)$, which in practice is not very convenient. Therefore time-invariant systems are widely used in DSP.

☐ **EXAMPLE 2.5** Determine whether the following systems are linear:

1. $y(n) = T[x(n)] = 3x^2(n)$
2. $y(n) = 2x(n-2) + 5$
3. $y(n) = x(n+1) - x(n-1)$

Solution Let $y_1(n) = T\big[x_1(n)\big]$ and $y_2(n) = T\big[x_2(n)\big]$. We will determine the response of each system to the linear combination $a_1x_1(n) + a_2x_2(n)$ and check whether it is equal to the linear combination $a_1x_1(n) + a_2x_2(n)$ where a_1 and a_2 are arbitrary constants.

1. $y(n) = T[x(n)] = 3x^2(n)$: Consider

$$\begin{aligned} T\big[a_1x_1(n) + a_2x_2(n)\big] &= 3\left[a_1x_1(n) + a_2x_2(n)\right]^2 \\ &= 3a_1^2x_1^2(n) + 3a_2^2x_2^2(n) + 6a_1a_2x_1(n)x_2(n) \end{aligned}$$

which is not equal to

$$a_1y_1(n) + a_2y_2(n) = 3a_1^2x_1^2(n) + 3a_2^2x_2^2(n)$$

Hence the given system is nonlinear.

2. $y(n) = 2x(n-2) + 5$: Consider

$$\begin{aligned} T\big[a_1x_1(n) + a_2x_2(n)\big] &= 2\left[a_1x_1(n-2) + a_2x_2(n-2)\right] + 5 \\ &= a_1y_1(n) + a_2y_2(n) - 5 \end{aligned}$$

Clearly, the given system is nonlinear even though the input-output relation is a straight-line function.

3. $y(n) = x(n+1) - x(1-n)$: Consider

$$\begin{aligned} T[a_1x_1(n) + a_2x_2(n)] &= a_1x_1(n+1) + a_2x_2(n+1) + a_1x_1(1-n) \\ &\quad + a_2x_2(1-n) \\ &= a_1[x_1(n+1) - x_1(1-n)] \\ &\quad + a_2[x_2(n+1) - x_2(1-n)] \\ &= a_1y_1(n) + a_2y_2(n) \end{aligned}$$

Hence the given system is linear. □

Linear time-invariant (LTI) system A linear system in which an input-output pair, $x(n)$ and $y(n)$, is invariant to a shift k in time is called a linear time-invariant system i.e.,

$$y(n) = L[x(n)] \Rightarrow L[x(n-k)] = y(n-k) \tag{2.12}$$

For an LTI system the $L[\cdot]$ and the shifting operators are reversible as shown here.

$$\begin{tikzcd} x(n) \arrow[r] & \boxed{L[\cdot]} \arrow[r] & y(n) \arrow[r] & \boxed{\text{Shift by } k} \arrow[r] & y(n-k) \end{tikzcd}$$

$$\begin{tikzcd} x(n) \arrow[r] & \boxed{\text{Shift by } k} \arrow[r] & x(n-k) \arrow[r] & \boxed{L[\cdot]} \arrow[r] & y(n-k) \end{tikzcd}$$

□ **EXAMPLE 2.6** Determine whether the following linear systems are time-invariant.

1. $y(n) = L[x(n)] = 10\sin(0.1\pi n)x(n)$
2. $y(n) = L[x(n)] = x(n+1) - x(1-n)$
3. $y(n) = L[x(n)] = \frac{1}{4}x(n) + \frac{1}{2}x(n-1) + \frac{1}{4}x(n-2)$

Solution First we will compute the response $y_k(n) \triangleq L[x(n-k)]$ to the shifted input sequence. This is obtained by subtracting k from the arguments of every input sequence term on the right-hand side of the linear transformation. To determine time-invariance, we will then compare it to the shifted output sequence $y(n-k)$, obtained after replacing every n by $(n-k)$ on the right-hand side of the linear transformation.

1. $y(n) = L[x(n)] = 10\sin(0.1\pi n)x(n)$: The response due to shifted input is

$$y_k(n) = L[x(n-k)] = 10\sin(0.1\pi n)x(n-k)$$

while the shifted output is

$$y(n-k) = 10\sin[0.1\pi(n-k)]x(n-k) \neq y_k(n).$$

Hence the given system is not time-invariant.

2. $y(n) = L[x(n)] = x(n+1) - x(1-n)$: The response due to shifted input is

$$y_k(n) = L[x(n-k)] = x(n-k) - x(1-n-k)$$

while the shifted output is

$$y(n-k) = x(n-k) - x(1-[n-k]) = x(n-k) - x(1-n+k) \neq y_k(n).$$

Hence the given system is not time-invariant.

3. $y(n) = L[x(n)] = \frac{1}{4}x(n) + \frac{1}{2}x(n-1) + \frac{1}{4}x(n-2)$: The response due to shifted input is

$$y_k(n) = L[x(n-k)] = \tfrac{1}{4}x(n-k) + \tfrac{1}{2}x(n-1-k) + \tfrac{1}{4}x(n-2-k)$$

while the shifted output is

$$y(n-k) = \tfrac{1}{4}x(n-k) + \tfrac{1}{2}x(n-k-1) + \tfrac{1}{4}x(n-k-2) = y_k(n)$$

Hence the given system is time-invariant. □

We will denote an LTI system by the operator $LTI\,[\cdot]$. Let $x(n)$ and $y(n)$ be the input-output pair of an LTI system. Then the time-varying function $h(n,k)$ becomes a time-invariant function $h(n-k)$, and the output from (2.11) is given by

$$y(n) = LTI\,[x(n)] = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \tag{2.13}$$

The impulse response of an LTI system is given by $h(n)$. The mathematical operation in (2.13) is called a *linear convolution sum* and is denoted by

$$y(n) \overset{\triangle}{=} x(n) * h(n) \tag{2.14}$$

Hence an LTI system is completely characterized in the time domain by the impulse response $h(n)$.

$$x(n) \longrightarrow \boxed{h(n)} \longrightarrow y(n) = x(n) * h(n)$$

We will explore several properties of the convolution in Problem P2.14.

Stability This is a very important concept in linear system theory. The primary reason for considering stability is to avoid building harmful systems or to avoid burnout or saturation in the system operation. A system is said to be *bounded-input bounded-output (BIBO) stable* if every bounded input produces a bounded output.

$$|x(n)| < \infty \Rightarrow |y(n)| < \infty, \forall x, y$$

An LTI system is BIBO stable if and only if its impulse response is *absolutely summable.*

$$\text{BIBO Stability} \Longleftrightarrow \sum_{-\infty}^{\infty} |h(n)| < \infty \tag{2.15}$$

Causality This important concept is necessary to make sure that systems can be built. A system is said to be causal if the output at index n_0 depends only on the input up to and including the index n_0; that is, the output does not depend on the future values of the input. An LTI system is causal if and only if the impulse response

$$h(n) = 0, \quad n < 0 \tag{2.16}$$

Such a sequence is termed a *causal sequence.* In signal processing, unless otherwise stated, we will always assume that the system is causal.

2.3 CONVOLUTION

We introduced the convolution operation (2.14) to describe the response of an LTI system. In DSP it is an important operation and has many other uses that we will see throughout this book. Convolution can be evaluated in many different ways. If the sequences are mathematical functions (of finite or infinite duration), then we can analytically evaluate (2.14) for all n to obtain a functional form of $y(n)$.

□ **EXAMPLE 2.7** Let the rectangular pulse $x(n) = u(n) - u(n-10)$ of Example 2.4 be an input to an LTI system with impulse response

$$h(n) = (0.9)^n \, u(n)$$

Determine the output $y(n)$.

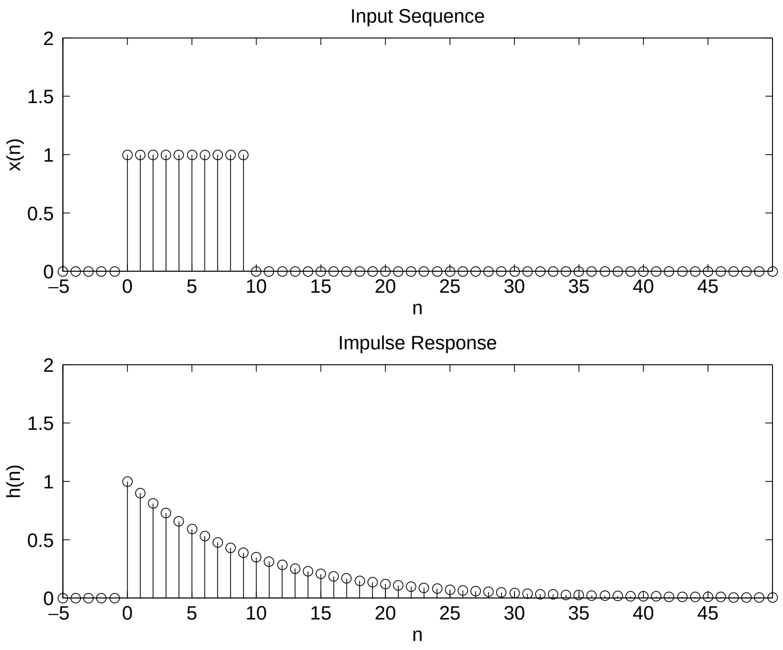


FIGURE 2.5 *The input sequence and the impulse response in Example 2.7*

Solution The input $x(n)$ and the impulse response $h(n)$ are shown in Figure 2.5. From (2.14)

$$y(n) = \sum_{k=0}^{9}(1)\,(0.9)^{(n-k)}\,u(n-k) = (0.9)^n \sum_{k=0}^{9}(0.9)^{-k}\,u(n-k) \qquad \textbf{(2.17)}$$

The sum in (2.17) is almost a geometric series sum except that the term $u(n-k)$ takes different values depending on n and k. There are three possible conditions under which $u(n-k)$ can be evaluated.

CASE i $n < 0$: Then $u(n-k) = 0, \quad 0 \le k \le 9$. Hence from (2.17)

$$y(n) = 0 \qquad \textbf{(2.18)}$$

CASE ii In this case the nonzero values of $x(n)$ and $h(n)$ *do not overlap.*
$0 \le n < 9$: Then $u(n-k) = 1$, $0 \le k \le n$. Hence from (2.17)

$$y(n) = (0.9)^n \sum_{k=0}^{n}(0.9)^{-k} = (0.9)^n \sum_{k=0}^{n}[(0.9)^{-1}]^k$$

$$= (0.9)^n \frac{1-(0.9)^{-(n+1)}}{1-(0.9)^{-1}} = 10[1-(0.9)^{n+1}], \quad 0 \le n < 9 \qquad \textbf{(2.19)}$$

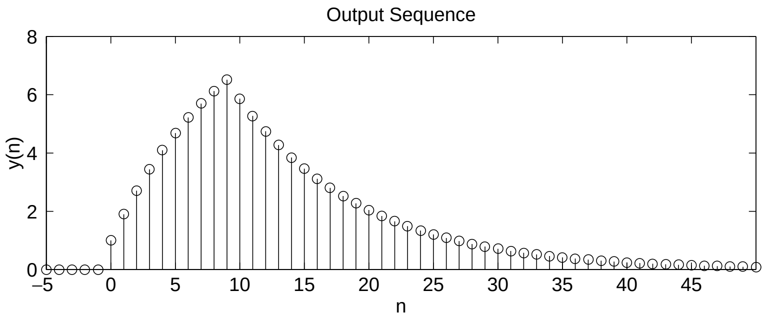


FIGURE 2.6 *The output sequence in Example 2.7*

CASE iii In this case the impulse response $h(n)$ *partially overlaps* the input $x(n)$.
$n \geq 9$: Then $u(n-k) = 1$, $0 \leq k \leq 9$ and from (2.17)

$$\begin{aligned} y(n) &= (0.9)^n \sum_{k=0}^{9} (0.9)^{-k} \\ &= (0.9)^n \frac{1-(0.9)^{-10}}{1-(0.9)^{-1}} = 10(0.9)^{n-9}[1-(0.9)^{10}], \quad n \geq 9 \end{aligned} \tag{2.20}$$

In this last case $h(n)$ *completely overlaps* $x(n)$.

The complete response is given by (2.18), (2.19), and (2.20). It is shown in Figure 2.6 which depicts the distortion of the input pulse. □

This example can also be done using a method called graphical convolution, in which (2.14) is given a graphical interpretation. In this method, $h(n-k)$ is interpreted as a *folded-and-shifted* version of $h(k)$. The output $y(n)$ is obtained as a sample sum under the overlap of $x(k)$ and $h(n-k)$. We use an example to illustrate this.

□ **EXAMPLE 2.8** Given the following two sequences

$$x(n) = [3, 11, \underset{\uparrow}{7}, 0, -1, 4, 2], \quad -3 \leq n \leq 3; \qquad h(n) = [2, \underset{\uparrow}{3}, 0, -5, 2, 1], \quad -1 \leq n \leq 4$$

determine the convolution $y(n) = x(n) * h(n)$.

Solution In Figure 2.7 we show four plots. The top-left plot shows $x(k)$ and $h(k)$, the original sequences. The top-right plot shows $x(k)$ and $h(-k)$, the folded version of $h(k)$. The bottom-left plot shows $x(k)$ and $h(-1-k)$, the folded-and-shifted-by- -1 version of $h(k)$. Then

$$\sum_k x(k)h(-1-k) = 3 \times (-5) + 11 \times 0 + 7 \times 3 + 0 \times 2 = 6 = y(-1)$$

The bottom-right plot shows $x(k)$ and $h(2-k)$, the folded-and-shifted-by-2 version of $h(k)$, which gives

$$\sum_k x(k)h(2-k) = 11 \times 1 + 7 \times 2 + 0 \times (-5) + (-1) \times 0 + 4 \times 3 + 2 \times 2 = 41 = y(2)$$

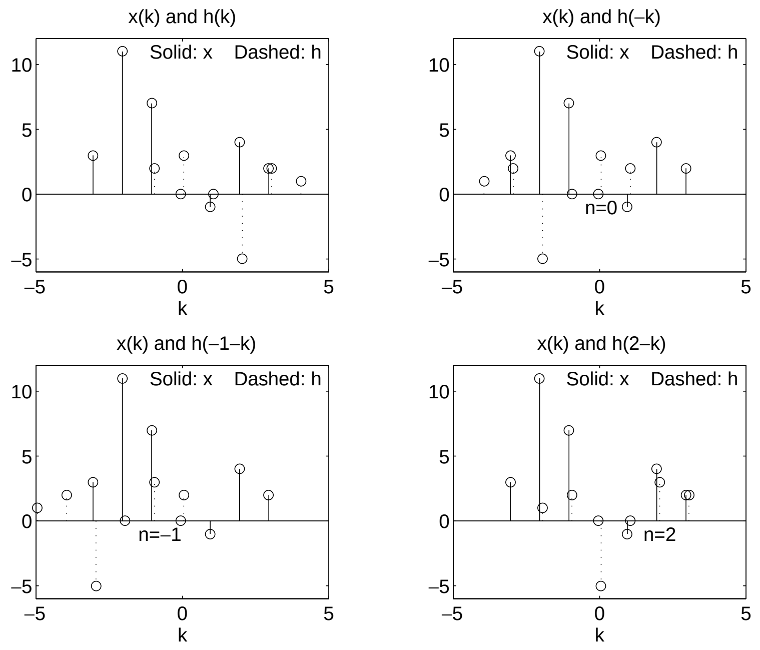


FIGURE 2.7 *Graphical convolution in Example 2.8*

Thus we have obtained two values of $y(n)$. Similar graphical calculations can be done for other remaining values of $y(n)$. Note that the beginning point (first nonzero sample) of $y(n)$ is given by $n = -3 + (-1) = -4$, while the end point (the last nonzero sample) is given by $n = 3 + 4 = 7$. The complete output is given by

$$y(n) = \{6, 31, 47, 6, \underset{\uparrow}{-51}, -5, 41, 18, -22, -3, 8, 2\}$$

Students are strongly encouraged to verify the above result. Note that the resulting sequence $y(n)$ has a *longer length* than both the $x(n)$ and $h(n)$ sequences.

□

2.3.1 MATLAB IMPLEMENTATION

If arbitrary sequences are of infinite duration, then MATLAB cannot be used *directly* to compute the convolution. MATLAB does provide a built-in function called `conv` that computes the convolution between two finite-duration sequences. The `conv` function assumes that the two sequences begin at $n = 0$ and is invoked by

```
>> y = conv(x,h);
```

For example, to do the convolution in Example 2.7, we could use

```
>> x = [3, 11, 7, 0, -1, 4, 2];  h = [2, 3, 0, -5, 2, 1];
>> y = conv(x, h)
y =
  6    31    47    6    -51   -5    41    18    -22    -3    8    2
```

to obtain the correct $y(n)$ values. However, the `conv` function neither provides nor accepts any timing information if the sequences have arbitrary support. What is needed is a beginning point and an end point of $y(n)$. Given finite duration $x(n)$ and $h(n)$, it is easy to determine these points. Let

$$\{x(n);\ n_{xb} \le n \le n_{xe}\} \qquad \text{and} \qquad \{h(n);\ n_{hb} \le n \le n_{he}\}$$

be two finite-duration sequences. Then referring to Example 2.8 we observe that the beginning and end points of $y(n)$ are

$$n_{yb} = n_{xb} + n_{hb} \quad \text{and} \quad n_{ye} = n_{xe} + n_{he}$$

respectively. A simple modification of the `conv` function, called `conv_m`, which performs the convolution of arbitrary support sequences can now be designed.

```
function [y,ny] = conv_m(x,nx,h,nh)
% Modified convolution routine for signal processing
% -----------------------------------------------------
%  [y,ny] = conv_m(x,nx,h,nh)
%  [y,ny] = convolution result
%  [x,nx] = first signal
%  [h,nh] = second signal
%
nyb = nx(1)+nh(1); nye = nx(length(x)) + nh(length(h));
ny = [nyb:nye];   y = conv(x,h);
```

□ **EXAMPLE 2.9** Perform the convolution in Example 2.8 using the `conv_m` function.

Solution MATLAB script:

```
>> x = [3, 11, 7, 0, -1, 4, 2]; nx = [-3:3];
>> h = [2, 3, 0, -5, 2, 1]; ny = [-1:4];
```

```
>> [y,ny] = conv_m(x,nx,h,nh)
y =
  6    31    47     6    -51   -5    41    18    -22   -3    8    2
ny =
 -4    -3    -2    -1     0     1     2     3     4     5    6    7
```

Hence

$$y(n) = \{6, 31, 47, 6, \underset{\uparrow}{-51}, -5, 41, 18, -22, -3, 8, 2\}$$

as in Example 2.8. □

An alternate method in MATLAB can be used to perform the convolution. This method uses a matrix-vector multiplication approach, which we will explore in Problem P2.17.

2.3.2 SEQUENCE CORRELATIONS REVISITED

If we compare the convolution operation (2.14) with that of the crosscorrelation of two sequences defined in (2.8), we observe a close resemblance. The crosscorrelation $r_{yx}(\ell)$ can be put in the form

$$r_{yx}(\ell) = y(\ell) * x(-\ell)$$

with the autocorrelation $r_{xx}(\ell)$ in the form

$$r_{xx}(\ell) = x(\ell) * x(-\ell)$$

Therefore these correlations can be computed using the `conv_m` function if sequences are of finite duration.

□ **EXAMPLE 2.10** In this example we will demonstrate one application of the crosscorrelation sequence. Let

$$x(n) = [3, 11, 7, \underset{\uparrow}{0}, -1, 4, 2]$$

be a prototype sequence, and let $y(n)$ be its noise-corrupted-and-shifted version

$$y(n) = x(n-2) + w(n)$$

where $w(n)$ is Gaussian sequence with mean 0 and variance 1. Compute the crosscorrelation between $y(n)$ and $x(n)$.

Solution From the construction of $y(n)$ it follows that $y(n)$ is "similar" to $x(n-2)$ and hence their crosscorrelation would show the strongest similarity at $\ell = 2$. To test this out using MATLAB, let us compute the crosscorrelation using two different noise sequences.

```
% noise sequence 1
>> x = [3, 11, 7, 0, -1, 4, 2]; nx=[-3:3]; % given signal x(n)
>> [y,ny] = sigshift(x,nx,2);              % obtain x(n-2)
>> w = randn(1,length(y)); nw = ny;        % generate w(n)
>> [y,ny] = sigadd(y,ny,w,nw);             % obtain y(n) = x(n-2) + w(n)
>> [x,nx] = sigfold(x,nx);                 % obtain x(-n)
```

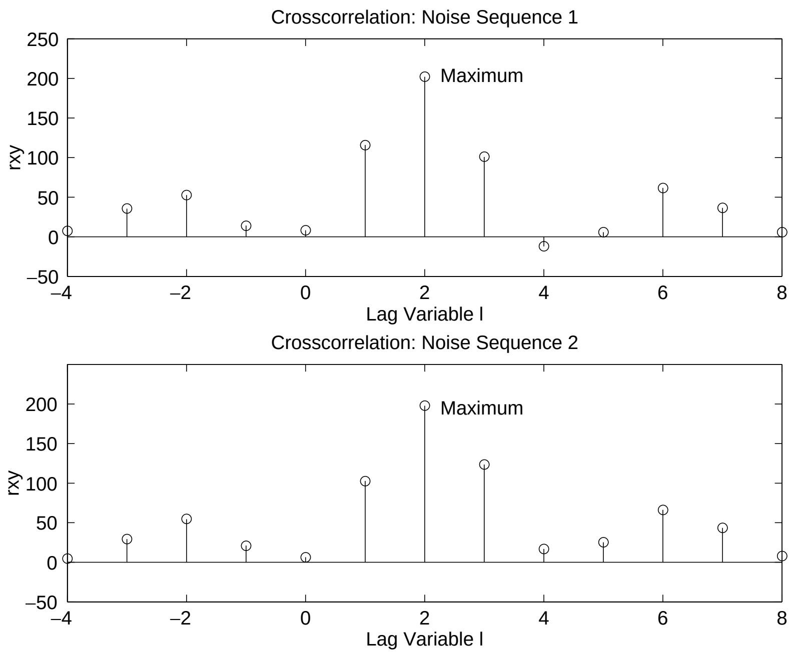


FIGURE 2.8 *Crosscorrelation sequence with two different noise realizations*

```
>> [rxy,nrxy] = conv_m(y,ny,x,nx);         % crosscorrelation
>> subplot(1,1,1), subplot(2,1,1);stem(nrxy,rxy)
>> axis([-5,10,-50,250]);xlabel('lag variable l')
>> ylabel('rxy');title('Crosscorrelation: noise sequence 1')
%
% noise sequence 2
>> x = [3, 11, 7, 0, -1, 4, 2]; nx=[-3:3]; % given signal x(n)
>> [y,ny] = sigshift(x,nx,2);              % obtain x(n-2)
>> w = randn(1,length(y)); nw = ny;        % generate w(n)
>> [y,ny] = sigadd(y,ny,w,nw);             % obtain y(n) = x(n-2) + w(n)
>> [x,nx] = sigfold(x,nx);                 % obtain x(-n)
>> [rxy,nrxy] = conv_m(y,ny,x,nx);         % crosscorrelation
>> subplot(2,1,2);stem(nrxy,rxy)
>> axis([-5,10,-50,250]);xlabel('lag variable l')
>> ylabel('rxy');title('Crosscorrelation: noise sequence 2')
```

From Figure 2.8 we observe that the crosscorrelation indeed peaks at $\ell = 2$, which implies that $y(n)$ is similar to $x(n)$ shifted by 2. This approach can be used in applications like radar signal processing in identifying and localizing targets. □

Note that the signal-processing toolbox in MATLAB also provides a function called `xcorr` for sequence correlation computations. In its simplest form

```
>> xcorr(x,y)
```

computes the crosscorrelation between vectors `x` and `y`, while

```
>> xcorr(x)
```

computes the autocorrelation of vector `x`. It generates results that are identical to the one obtained from the proper use of the `conv_m` function. However, the `xcorr` function cannot provide the timing (or lag) information (as done by the `conv_m` function), which then must be obtained by some other means.

2.4 DIFFERENCE EQUATIONS

An LTI discrete system can also be described by a linear constant coefficient difference equation of the form

$$\sum_{k=0}^{N} a_k y(n-k) = \sum_{m=0}^{M} b_m x(n-m), \quad \forall n \tag{2.21}$$

If $a_N \neq 0$, then the difference equation is of order N. This equation describes a recursive approach for computing the current output, given the input values and previously computed output values. In practice this equation is computed forward in time, from $n = -\infty$ to $n = \infty$. Therefore another form of this equation is

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) - \sum_{k=1}^{N} a_k y(n-k) \tag{2.22}$$

A solution to this equation can be obtained in the form

$$y(n) = y_H(n) + y_P(n)$$

The *homogeneous part* of the solution, $y_H(n)$, is given by

$$y_H(n) = \sum_{k=1}^{N} c_k z_k^n$$

where $z_k, k = 1, \dots, N$ are N roots (also called *natural frequencies*) of the characteristic equation

$$\sum_0^N a_k z^k = 0$$

This characteristic equation is important in determining the stability of systems. If the roots z_k satisfy the condition

$$|z_k| < 1, \; k = 1, \dots, N \tag{2.23}$$

then a causal system described by (2.22) is stable. The *particular part* of the solution, $y_P(n)$, is determined from the right-hand side of (2.21). In Chapter 4 we will discuss the analytical approach of solving difference equations using the z-transform.

2.4.1 MATLAB IMPLEMENTATION

A function called `filter` is available to solve difference equations numerically, given the input and the difference equation coefficients. In its simplest form this function is invoked by

```
y = filter(b,a,x)
```

where

```
b = [b0, b1, ..., bM]; a = [a0, a1, ..., aN];
```

are the coefficient arrays from the equation given in (2.21), and `x` is the input sequence array. The output `y` has the same length as input `x`. One must ensure that the coefficient `a0` not be zero.

To compute and plot impulse response, MATLAB provides the function `impz`. When invoked by

```
h = impz(b,a,n);
```

it computes samples of the impulse response of the filter at the sample indices given in `n` with numerator coefficients in `b` and denominator coefficients in `a`. When no output arguments are given, the `impz` function plots the response in the current figure window using the `stem` function. We will illustrate the use of these functions in the following example.

□ **EXAMPLE 2.11** Given the following difference equation

$$y(n) - y(n-1) + 0.9y(n-2) = x(n); \quad \forall n$$

a. Calculate and plot the impulse response $h(n)$ at $n = -20, \dots, 100$.
b. Calculate and plot the unit step response $s(n)$ at $n = -20, \dots, 100$.
c. Is the system specified by $h(n)$ stable?

Solution From the given difference equation the coefficient arrays are

```
b = [1]; a=[1, -1, 0.9];
```

a. MATLAB script:

```
>> b = [1]; a = [1, -1, 0.9];  n = [-20:120];
>> h = impz(b,a,n);
>> subplot(2,1,1); stem(n,h);
>> title('Impulse Response'); xlabel('n'); ylabel('h(n)')
```

The plot of the impulse response is shown in Figure 2.9.

b. MATLAB script:

```
>> x = stepseq(0,-20,120);   s = filter(b,a,x);
>> subplot(2,1,2); stem(n,s)
>> title('Step Response'); xlabel('n'); ylabel('s(n)')
```

The plot of the unit step response is shown in Figure 2.9.

c. To determine the stability of the system, we have to determine $h(n)$ for all n. Although we have not described a method to solve the difference equation,

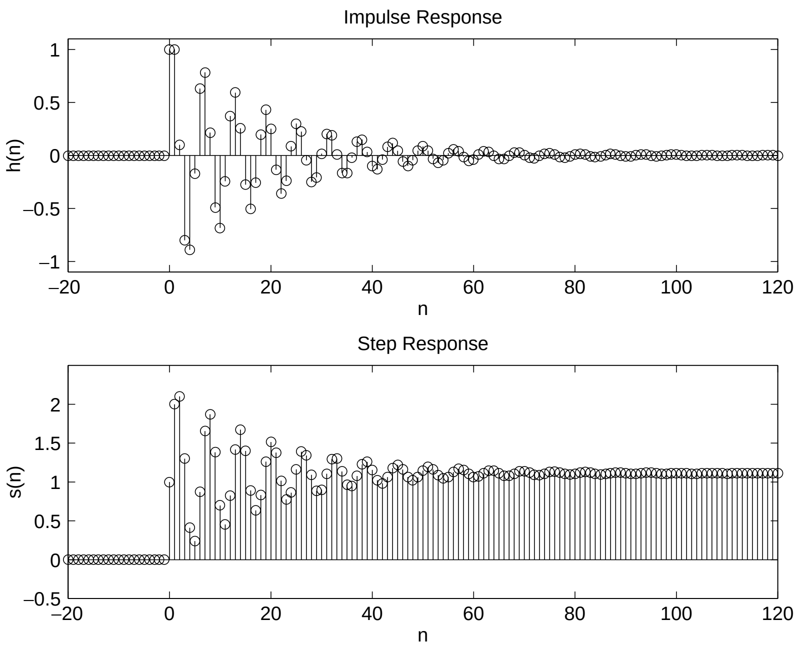


FIGURE 2.9 *Impulse response and step response plots in Example 2.11*

we can use the plot of the impulse response to observe that $h(n)$ is practically zero for $n > 120$. Hence the sum $\sum |h(n)|$ can be determined from MATLAB using

```
>> sum(abs(h))
ans = 14.8785
```

which implies that the system is stable. An alternate approach is to use the stability condition (2.23) using MATLAB's `roots` function.

```
>>z = roots(a);   magz = abs(z)
magz =  0.9487
        0.9487
```

Since the magnitudes of both roots are less than one, the system is stable.

□

In the previous section we noted that if one or both sequences in the convolution are of infinite length, then the `conv` function cannot be used. If one of the sequences is of infinite length, then it is possible to use MATLAB for numerical evaluation of the convolution. This is done using the `filter` function as we will see in the following example.

□ **EXAMPLE 2.12** Let us consider the convolution given in Example 2.7. The input sequence is of finite duration

$$x(n) = u(n) - u(n-10)$$

while the impulse response is of infinite duration

$$h(n) = (0.9)^n\, u(n)$$

Determine $y(n) = x(n) * h(n)$.

Solution If the LTI system, given by the impulse response $h(n)$, can be described by a difference equation, then $y(n)$ can be obtained from the `filter` function. From the $h(n)$ expression

$$(0.9)\, h(n-1) = (0.9)\, (0.9)^{n-1}\, u(n-1) = (0.9)^n\, u(n-1)$$

or

$$\begin{aligned} h(n) - (0.9)\, h(n-1) &= (0.9)^n\, u(n) - (0.9)^n\, u(n-1) \\ &= (0.9)^n\, [u(n) - u(n-1)] = (0.9)^n\, \delta(n) \\ &= \delta(n) \end{aligned}$$

The last step follows from the fact that $\delta(n)$ is nonzero only at $n = 0$. By definition $h(n)$ is the output of an LTI system when the input is $\delta(n)$. Hence substituting $x(n)$ for $\delta(n)$ and $y(n)$ for $h(n)$, the difference equation is

$$y(n) - 0.9y(n-1) = x(n)$$

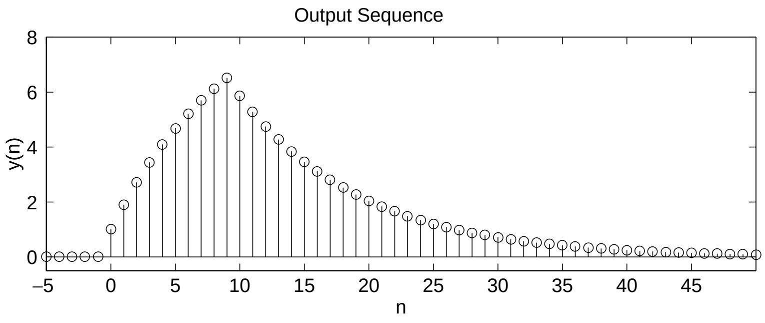


FIGURE 2.10 *Output sequence in Example 2.12*

Now MATLAB's `filter` function can be used to compute the convolution indirectly.

```
>> b = [1]; a = [1,-0.9];
>> n = -5:50; x = stepseq(0,-5,50) - stepseq(10,-5,50);
>> y = filter(b,a,x);
>> subplot(2,1,2); stem(n,y); title('Output sequence')
>> xlabel('n'); ylabel('y(n)'); axis([-5,50,-0.5,8])
```

The plot of the output is shown in Figure 2.10, which is exactly the same as that in Figure 2.6. □

In Example 2.12 the impulse response was a one-sided exponential sequence for which we could determine a difference equation representation. This means that not all infinite-length impulse responses can be converted into difference equations. The above analysis, however, can be extended to a linear combination of one-sided exponential sequences, which results in higher-order difference equations. We will discuss this topic of conversion from one representation to another one in Chapter 4.

2.4.2 ZERO-INPUT AND ZERO-STATE RESPONSES

In digital signal processing the difference equation is generally solved forward in time from $n = 0$. Therefore initial conditions on $x(n)$ and $y(n)$ are necessary to determine the output for $n \geq 0$. The difference equation is then given by

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) - \sum_{k=1}^{N} a_k y(n-k);\ n \geq 0 \tag{2.24}$$

subject to the initial conditions:

$$\{y(n);\ -N \le n \le -1\} \qquad \text{and} \qquad \{x(n);\ -M \le n \le -1\}$$

A solution to (2.24) can be obtained in the form

$$y(n) = y_{ZI}(n) + y_{ZS}(n)$$

where $y_{ZI}(n)$ is called the *zero-input* solution, which is a solution due to the initial conditions alone (assuming they exist), while the *zero-state* solution, $y_{ZS}(n)$, is a solution due to input $x(n)$ alone (or assuming that the initial conditions are zero). In MATLAB another form of the function `filter` can be used to solve for the difference equation, given its initial conditions. We will illustrate the use of this form in Chapter 4.

2.4.3 DIGITAL FILTERS

Filter is a generic name that means a linear time-invariant system designed for a specific job of frequency selection or frequency discrimination. Hence discrete-time LTI systems are also called digital filters. There are two types of digital filters.

FIR filter If the unit impulse response of an LTI system is of finite duration, then the system is called a *finite-duration impulse response* (or FIR) filter. Hence for an FIR filter $h(n) = 0$ for $n < n_1$ and for $n > n_2$. The following part of the difference equation (2.21) describes a *causal* FIR filter:

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) \tag{2.25}$$

Furthermore, $h(0) = b_0$, $h(1) = b_1, \ldots, h(M) = b_M$, while all other $h(n)$'s are 0. FIR filters are also called *nonrecursive* or *moving average* (MA) filters. In MATLAB FIR filters are represented either as impulse response values $\{h(n)\}$ or as difference equation coefficients $\{b_m\}$ and $\{a_0 = 1\}$. Therefore to implement FIR filters, we can use either the `conv(x,h)` function (and its modification that we discussed) or the `filter(b,1,x)` function. There is a difference in the outputs of these two implementations that should be noted. The output sequence from the `conv(x,h)` function has a *longer length* than both the $x(n)$ and $h(n)$ sequences. On the other hand, the output sequence from the `filter(b,1,x)` function has exactly the *same length* as the input $x(n)$ sequence. In practice (and especially for processing signals) the use of the `filter` function is encouraged.

IIR filter If the impulse response of an LTI system is of infinite duration, then the system is called an *infinite-duration impulse response* (or IIR) filter. The following part of the difference equation (2.21):

$$\sum_{k=0}^{N} a_k y(n-k) = x(n) \tag{2.26}$$

describes a *recursive* filter in which the output $y(n)$ is recursively computed from its previously computed values and is called an *autoregressive* (AR) filter. The impulse response of such filter is of infinite duration and hence it represents an IIR filter. The general equation (2.21) also describes an IIR filter. It has two parts: an AR part and an MA part. Such an IIR filter is called an *autoregressive moving average*, or an ARMA, filter. In MATLAB, IIR filters are described by the difference equation coefficients $\{b_m\}$ and $\{a_k\}$ and are implemented by the `filter(b,a,x)` function.

2.5 PROBLEMS

P2.1 Generate the following sequences using the basic MATLAB signal functions and the basic MATLAB signal operations discussed in this chapter. Plot signal samples using the `stem` function.

1. $x_1(n) = 3\delta(n+2) + 2\delta(n) - \delta(n-3) + 5\delta(n-7)$, $-5 \le n \le 15$.
2. $x_2(n) = \sum_{k=-5}^{5} e^{-|k|}\delta(n-2k)$, $-10 \le n \le 10$.
3. $x_3(n) = 10u(n) - 5u(n-5) - 10u(n-10) + 5u(n-15)$.
4. $x_4(n) = e^{0.1n}[u(n+20) - u(n-10)]$.
5. $x_5(n) = 5[\cos(0.49\pi n) + \cos(0.51\pi n)]$, $-200 \le n \le 200$. Comment on the waveform shape.
6. $x_6(n) = 2\sin(0.01\pi n)\cos(0.5\pi n)$, $-200 \le n \le 200$. Comment on the waveform shape.
7. $x_7(n) = e^{-0.05n}\sin(0.1\pi n + \pi/3)$, $0 \le n \le 100$. Comment on the waveform shape.
8. $x_8(n) = e^{0.01n}\sin(0.1\pi n)$, $0 \le n \le 100$. Comment on the waveform shape.

P2.2 Generate the following random sequences and obtain their histogram using the `hist` function with 100 bins. Use the `bar` function to plot each histogram.

1. $x_1(n)$ is a random sequence whose samples are independent and uniformly distributed over $[0,2]$ interval. Generate 100,000 samples.
2. $x_2(n)$ is a Gaussian random sequence whose samples are independent with mean 10 and variance 10. Generate 10,000 samples.
3. $x_3(n) = x_1(n) + x_1(n-1)$ where $x_1(n)$ is the random sequence given in part 1 above. Comment on the shape of this histogram and explain the shape.
4. $x_4(n) = \sum_{k=1}^{4} y_k(n)$ where each random sequence $y_k(n)$ is independent of others with samples uniformly distributed over $[-0.5, 0.5]$. Comment on the shape of this histogram.

P2.3 Generate the following periodic sequences and plot their samples (using the `stem` function) over the indicated number of periods.

1. $\tilde{x}_1(n) = \{\ldots, -2, -1, \underset{\uparrow}{0}, 1, 2, \ldots\}_{\text{periodic}}$. Plot 5 periods.
2. $\tilde{x}_2(n) = e^{0.1n}[u(n) - u(n-20]_{\text{periodic}}$. Plot 3 periods.
3. $\tilde{x}_3(n) = sin(0.1\pi n)[u(n) - u(n-10)]$. Plot 4 periods.
4. $\tilde{x}_4(n) = \{\ldots, \underset{\uparrow}{1}, 2, 3, \ldots\}_{\text{periodic}} + \{\ldots, \underset{\uparrow}{1}, 2, 3, 4, \ldots\}_{\text{periodic}}$, $0 \leq n \leq 24$. What is the period of $\tilde{x}_4(n)$?

P2.4 Let $x(n) = \{2, 4, -3, \underset{\uparrow}{1}, -5, 4, 7\}$. Generate and plot the samples (use the `stem` function) of the following sequences.

1. $x_1(n) = 2x(n-3) + 3x(n+4) - x(n)$
2. $x_2(n) = 4x(4+n) + 5x(n+5) + 2x(n)$
3. $x_3(n) = x(n+3)x(n-2) + x(1-n)x(n+1)$
4. $x_4(n) = 2e^{0.5n}x(n) + \cos(0.1\pi n)\, x(n+2)\,, \; -10 \leq n \leq 10$

P2.5 The complex exponential sequence $e^{j\omega_0 n}$ or the sinusoidal sequence $\cos(\omega_0 n)$ are periodic if the *normalized* frequency $f_0 \triangleq \dfrac{\omega_0}{2\pi}$ is a rational number; that is, $f_0 = \dfrac{K}{N}$, where K and N are integers.

1. Prove the above result.
2. Generate $\exp(0.1\pi n)$, $-100 \leq n \leq 100$. Plot its real and imaginary parts using the `stem` function. Is this sequence periodic? If it is, what is its fundamental period? From the examination of the plot what interpretation can you give to the integers K and N above?
3. Generate and plot $\cos(0.1n)$, $-20 \leq n \leq 20$. Is this sequence periodic? What do you conclude from the plot? If necessary examine the values of the sequence in MATLAB to arrive at your answer.

P2.6 Using the `evenodd` function, decompose the following sequences into their even and odd components. Plot these components using the `stem` function.

1. $x_1(n) = \{\underset{\uparrow}{0}, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
2. $x_2(n) = e^{0.1n}[u(n+5) - u(n-10)]$.
3. $x_3(n) = \cos(0.2\pi n + \pi/4)$, $-20 \leq n \leq 20$.
4. $x_4(n) = e^{-0.05n}\sin(0.1\pi n + \pi/3)$, $0 \leq n \leq 100$.

P2.7 A complex-valued sequence $x_e(n)$ is called *conjugate-symmetric* if $x_e(n) = x_e^*(-n)$ and a complex-valued sequence $x_o(n)$ is called *conjugate-antisymmetric* if $x_o(n) = -x_o^*(-n)$. Then, any arbitrary complex-valued sequence $x(n)$ can be decomposed into $x(n) = x_e(n) + x_o(n)$ where $x_e(n)$ and $x_o(n)$ are given by

$$x_e(n) = \frac{1}{2}\left[x(n) + x^*(-n)\right] \qquad \text{and} \qquad x_o(n) = \frac{1}{2}\left[x(n) - x^*(-n)\right] \tag{2.27}$$

respectively.

1. Modify the `evenodd` function discussed in the text so that it accepts an arbitrary sequence and decomposes it into its conjugate-symmetric and conjugate-antisymmetric components by implementing (2.27).
2. Decompose the following sequence:

$$x(n) = 10 \exp([-0.1 + \jmath 0.2\pi]n), \quad 0 \leq n \leq 10$$

into its conjugate-symmetric and conjugate-antisymmetric components. Plot their real and imaginary parts to verify the decomposition. (Use the `subplot` function.)

P2.8 The operation of *signal dilation* (or *decimation* or *down-sampling*) is defined by

$$y(n) = x(nM)$$

in which the sequence $x(n)$ is down-sampled by an integer factor M. For example, if

$$x(n) = \{\ldots, -2, 4, \underset{\uparrow}{3}, -6, 5, -1, 8, \ldots\}$$

then the down-sampled sequences by a factor 2 are given by

$$y(n) = \{\ldots, -2, \underset{\uparrow}{3}, 5, 8, \ldots\}$$

1. Develop a MATLAB function `dnsample` that has the form

```
function [y,m] = dnsample(x,n,M)
% Downsample sequence x(n) by a factor M to obtain y(m)
```

to implement the above operation. Use the indexing mechanism of MATLAB with careful attention to the origin of the time axis $n = 0$.
2. Generate $x(n) = \sin(0.125\pi n), \quad -50 \leq n \leq 50$. Decimate $x(n)$ by a factor of 4 to generate $y(n)$. Plot both $x(n)$ and $y(n)$ using `subplot` and comment on the results.
3. Repeat the above using $x(n) = \sin(0.5\pi n), \quad -50 \leq n \leq 50$. Qualitatively discuss the effect of down-sampling on signals.

P2.9 Using the `conv_m` function, determine the autocorrelation sequence $r_{xx}(\ell)$ and the crosscorrelation sequence $r_{xy}(\ell)$ for the following sequences.

$$x(n) = (0.9)^n, \quad 0 \leq n \leq 20; \qquad y(n) = (0.8)^{-n}, \quad -20 \leq n \leq 0$$

Describe your observations of these results.

P2.10 In a certain concert hall, echoes of the original audio signal $x(n)$ are generated due to the reflections at the walls and ceiling. The audio signal experienced by the listener $y(n)$ is a combination of $x(n)$ and its echoes. Let

$$y(n) = x(n) + \alpha x(n-k)$$

where k is the amount of delay in samples and α is its relative strength. We want to estimate the delay using the correlation analysis.

1. Determine analytically the crosscorrelation $r_{yx}(\ell)$ in terms of the autocorrelation $r_{xx}(\ell)$.
2. Let $x(n) = \cos(0.2\pi n) + 0.5\cos(0.6\pi n)$, $\alpha = 0.1$, and $k = 50$. Generate 200 samples of $y(n)$ and determine its crosscorrelation. Can you obtain α and k by observing $r_{yx}(\ell)$?

P2.11 Consider the following discrete-time systems:

$$\begin{aligned}
&T_1[x(n)] = x(n)u(n) && T_2[x(n)] = x(n) + n\,x(n+1)\\
&T_3[x(n)] = x(n) + \frac{1}{2}x(n-2) - \frac{1}{3}x(n-3)x(2n) && T_4[x(n)] = \textstyle\sum_{k=-\infty}^{n+5} 2x(k)\\
&T_5[x(n)] = x(2n) && T_6[x(n)] = \text{round}[x(n)]
\end{aligned}$$

where round[·] denotes rounding to the nearest integer.

1. Use (2.10) to determine analytically whether these systems are linear.
2. Let $x_1(n)$ be a uniformly distributed random sequence between $[0, 1]$ over $0 \le n \le 100$, and let $x_2(n)$ be a Gaussian random sequence with mean 0 and variance 10 over $0 \le n \le 100$. Using these sequences, verify the linearity of these systems. Choose any values for constants a_1 and a_2 in (2.10). You should use several realizations of the above sequences to arrive at your answers.

P2.12 Consider the discrete-time systems given in Problem P2.11.

1. Use (2.12) to determine analytically whether these systems are time-invariant.
2. Let $x(n)$ be a Gaussian random sequence with mean 0 and variance 10 over $0 \le n \le 100$. Using this sequence, verify the time invariance of the above systems. Choose any values for sample shift k in (2.12). You should use several realizations of the above sequence to arrive at your answers.

P2.13 For the systems given in Problem P2.11, determine analytically their stability and causality.

P2.14 The linear convolution defined in (2.14) has several properties:

$$\begin{aligned}
x_1(n) * x_2(n) &= x_1(n) * x_2(n) && : \text{Commutation}\\
[x_1(n) * x_2(n)] * x_3(n) &= x_1(n) * [x_2(n) * x_3(n)] && : \text{Association}\\
x_1(n) * [x_2(n) + x_3(n)] &= x_1(n) * x_2(n) + x_1(n) * x_3(n) && : \text{Distribution}\\
x(n) * \delta(n - n_0) &= x(n - n_0) && : \text{Identity}
\end{aligned} \tag{2.28}$$

1. Analytically prove these properties.
2. Using the following three sequences, verify the above properties.

$$\begin{aligned}
x_1(n) &= \cos(\pi n/4)[u(n+5) - u(n-25)]\\
x_2(n) &= (0.9)^{-n}[u(n) - u(n-20)]\\
x_3(n) &= \text{round}[5w(n)],\ -10 \le n \le 10;\ \text{where } w(n) \text{ is uniform over } [-1, 1]
\end{aligned}$$

Use the `conv_m` function.

P2.15 Determine analytically the convolution $y(n) = x(n) * h(n)$ of the following sequences, and verify your answers using the `conv_m` function.

1. $x(n) = \{2, -4, 5, \underset{\uparrow}{3}, -1, -2, 6\}$, $h(n) = \{1, \underset{\uparrow}{-1}, 1, -1, 1\}$

2. $x(n) = \{1, 1, \underset{\uparrow}{0}, 1, 1\}$, $h(n) = \{1, -2, -3, \underset{\uparrow}{4}\}$
3. $x(n) = (1/4)^{-n}[u(n+1) - u(n-4)]$, $h(n) = u(n) - u(n-5)$
4. $x(n) = n/4[u(n) - u(n-6)]$, $h(n) = 2[u(n+2) - u(n-3)]$

P2.16 Let $x(n) = (0.8)^n u(n)$, $h(n) = (-0.9)^n u(n)$, and $y(n) = h(n) * x(n)$. Use 3 columns and 1 row of subplots for the following parts.

1. Determine $y(n)$ analytically. Plot first 51 samples of $y(n)$ using the `stem` function.
2. Truncate $x(n)$ and $h(n)$ to 26 samples. Use `conv` function to compute $y(n)$. Plot $y(n)$ using the `stem` function. Compare your results with those of part 1.
3. Using the `filter` function, determine the first 51 samples of $x(n) * h(n)$. Plot $y(n)$ using the `stem` function. Compare your results with those of parts 1 and 2.

P2.17 When the sequences $x(n)$ and $h(n)$ are of finite duration N_x and N_h, respectively, then their linear convolution (2.13) can also be implemented using *matrix-vector multiplication.* If elements of $y(n)$ and $x(n)$ are arranged in column vectors $\mathbf{x}$ and $\mathbf{y}$ respectively, then from (2.13) we obtain

$$\mathbf{y} = \mathbf{H}\mathbf{x}$$

where linear shifts in $h(n-k)$ for $n = 0, \ldots, N_h - 1$ are arranged as rows in the matrix $\mathbf{H}$. This matrix has an interesting structure and is called a *Toeplitz* matrix. To investigate this matrix, consider the sequences

$$x(n) = \{\underset{\uparrow}{1}, 2, 3, 4, 5\} \qquad \text{and} \qquad h(n) = \{\underset{\uparrow}{6}, 7, 8, 9\}$$

1. Determine the linear convolution $y(n) = h(n) * x(n)$.
2. Express $x(n)$ as a 5×1 column vector $\mathbf{x}$ and $y(n)$ as a 8×1 column vector $\mathbf{y}$. Now determine the 8×5 matrix $\mathbf{H}$ so that $\mathbf{y} = \mathbf{H}\mathbf{x}$.
3. Characterize the matrix $\mathbf{H}$. From this characterization can you give a definition of a Toeplitz matrix? How does this definition compare with that of time invariance?
4. What can you say about the first column and the first row of $\mathbf{H}$?

P2.18 MATLAB provides a function called `toeplitz` to generate a Toeplitz matrix, given the first row and the first column.

1. Using this function and your answer to Problem P2.17, part 4, develop another MATLAB function to implement linear convolution. The format of the function should be

```
function [y,H]=conv_tp(h,x)
% Linear Convolution using Toeplitz Matrix
% ----------------------------------------
% [y,H] = conv_tp(h,x)
% y = output sequence in column vector form
% H = Toeplitz matrix corresponding to sequence h so that y = Hx
% h = Impulse response sequence in column vector form
% x = input sequence in column vector form
```

2. Verify your function on the sequences given in Problem P2.17.

P2.19 A linear and time-invariant system is described by the difference equation

$$y(n) - 0.5y(n-1) + 0.25y(n-2) = x(n) + 2x(n-1) + x(n-3)$$

1. Using the `filter` function, compute and plot the impulse response of the system over $0 \le n \le 100$.
2. Determine the stability of the system from this impulse response.
3. If the input to this system is $x(n) = [5 + 3\cos(0.2\pi n) + 4\sin(0.6\pi n)]\,u(n)$, determine the response $y(n)$ over $0 \le n \le 200$ using the `filter` function.

P2.20 A "simple" *digital differentiator* is given by

$$y(n) = x(n) - x(n-1)$$

which computes a backward first-order difference of the input sequence. Implement this differentiator on the following sequences, and plot the results. Comment on the appropriateness of this simple differentiator.

1. $x(n) = 5\,[u(n) - u(n-20)]$: a rectangular pulse
2. $x(n) = n\,[u(n) - u(n-10)] + (20-n)\,[u(n-10) - u(n-20)]$: a triangular pulse
3. $x(n) = \sin\left(\dfrac{\pi n}{25}\right)[u(n) - u(n-100)]$: a sinusoidal pulse

CHAPTER 3

The Discrete-time Fourier Analysis

We have seen how a linear and time-invariant system can be represented using its response to the unit sample sequence. This response, called the *unit impulse response* $h(n)$, allows us to compute the system response to any arbitrary input $x(n)$ using the linear convolution:

$$x(n) \longrightarrow \boxed{h(n)} \longrightarrow y(n) = h(n) * x(n)$$

This convolution representation is based on the fact that any signal can be represented by a linear combination of scaled and delayed unit samples. Similarly, we can also represent any arbitrary discrete signal as a linear combination of basis signals introduced in Chapter 2. Each basis signal set provides a new signal representation. Each representation has some advantages and some disadvantages depending upon the type of system under consideration. However, when the system is linear and time-invariant, only one representation stands out as the most useful. It is based on the complex exponential signal set $\{e^{j\omega n}\}$ and is called the *discrete-time Fourier transform.*

3.1 THE DISCRETE-TIME FOURIER TRANSFORM (DTFT)

If $x(n)$ is absolutely summable, that is, $\sum_{-\infty}^{\infty} |x(n)| < \infty$, then its discrete-time Fourier transform is given by

$$X(e^{j\omega}) \overset{\triangle}{=} \mathcal{F}[x(n)] = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \tag{3.1}$$

The inverse discrete-time Fourier transform (IDTFT) of $X(e^{j\omega})$ is given by

$$x(n) \triangleq \mathcal{F}^{-1}[X(e^{j\omega})] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega \qquad \textbf{(3.2)}$$

The operator $\mathcal{F}[\cdot]$ transforms a discrete signal $x(n)$ into a complex-valued continuous function $X(e^{j\omega})$ of real variable ω, called a digital frequency, which is measured in radians/sample.

□ **EXAMPLE 3.1** Determine the discrete-time Fourier transform of $x(n) = (0.5)^n\, u(n)$.

Solution The sequence $x(n)$ is absolutely summable; therefore its discrete-time Fourier transform exists.

$$\begin{aligned} X(e^{j\omega}) &= \sum_{-\infty}^{\infty} x(n)e^{-j\omega n} = \sum_{0}^{\infty} (0.5)^n\, e^{-j\omega n} \\ &= \sum_{0}^{\infty}(0.5e^{-j\omega})^n = \frac{1}{1-0.5e^{-j\omega}} = \frac{e^{j\omega}}{e^{j\omega}-0.5} \end{aligned}$$

□

□ **EXAMPLE 3.2** Determine the discrete-time Fourier transform of the following finite-duration sequence:

$$x(n) = \{1, \underset{\uparrow}{2}, 3, 4, 5\}$$

Solution Using definition (3.1),

$$X(e^{j\omega}) = \sum_{-\infty}^{\infty} x(n)e^{-j\omega n} = e^{j\omega} + 2 + 3e^{-j\omega} + 4e^{-j2\omega} + 5e^{-j3\omega}$$

□

Since $X(e^{j\omega})$ is a complex-valued function, we will have to plot its magnitude and its angle (or the real and the imaginary part) with respect to ω separately to visually describe $X(e^{j\omega})$. Now ω is a real variable between $-\infty$ and ∞, which would mean that we can plot only a part of the $X(e^{j\omega})$ function using MATLAB. Using two important properties of the discrete-time Fourier transform, we can reduce this domain to the $[0, \pi]$ interval for real-valued sequences. We will discuss other useful properties of $X(e^{j\omega})$ in the next section.

3.1.1 TWO IMPORTANT PROPERTIES

We will state the following two properties without proof.

1. **Periodicity:** The discrete-time Fourier transform $X(e^{j\omega})$ is periodic in ω with period 2π.
$$X(e^{j\omega}) = X(e^{j[\omega+2\pi]})$$
Implication: We need only one period of $X(e^{j\omega})$ (i.e., $\omega \in [0, 2\pi]$, or $[-\pi, \pi]$, etc.) for analysis and not the whole domain $-\infty < \omega < \infty$.
2. **Symmetry:** For real-valued $x(n)$, $X(e^{j\omega})$ is conjugate symmetric.
$$X(e^{-j\omega}) = X^*(e^{j\omega})$$
or
$$\begin{aligned}\mathrm{Re}[X(e^{-j\omega})] &= \mathrm{Re}[X(e^{j\omega})] \quad \text{(even symmetry)}\\ \mathrm{Im}[X(e^{-j\omega})] &= -\,\mathrm{Im}[X(e^{j\omega})] \quad \text{(odd symmetry)}\\ |X(e^{-j\omega})| &= |X(e^{j\omega})| \quad \text{(even symmetry)}\\ \angle X(e^{-j\omega}) &= -\angle X(e^{j\omega}) \quad \text{(odd symmetry)}\end{aligned}$$
Implication: To plot $X(e^{j\omega})$, we now need to consider only a half period of $X(e^{j\omega})$. Generally, in practice this period is chosen to be $\omega \in [0, \pi]$.

3.1.2 MATLAB IMPLEMENTATION

If $x(n)$ is of infinite duration, then MATLAB cannot be used directly to compute $X(e^{j\omega})$ from $x(n)$. However, we can use it to evaluate the expression $X(e^{j\omega})$ over $[0, \pi]$ frequencies and then plot its magnitude and angle (or real and imaginary parts).

□ **EXAMPLE 3.3** Evaluate $X(e^{j\omega})$ in Example 3.1 at 501 equispaced points between $[0, \pi]$ and plot its magnitude, angle, real, and imaginary parts.

Solution MATLAB script:

```
>> w = [0:1:500]*pi/500;  % [0, pi]  axis divided into 501 points.
>> X = exp(j*w) ./ (exp(j*w) - 0.5*ones(1,501));
>> magX = abs(X); angX = angle(X);  realX = real(X); imagX = imag(X);
>> subplot(2,2,1); plot(w/pi,magX); grid
>> xlabel('frequency in pi units'); title('Magnitude Part'); ylabel('Magnitude')
>> subplot(2,2,3); plot(w/pi,angX); grid
>> xlabel('frequency in pi units'); title('Angle Part'); ylabel('Radians')
>> subplot(2,2,2); plot(w/pi,realX); grid
>> xlabel('frequency in pi units'); title('Real Part'); ylabel('Real')
>> subplot(2,2,4); plot(w/pi,imagX); grid
>> xlabel('frequency in pi units'); title('Imaginary Part'); ylabel('Imaginary')
```

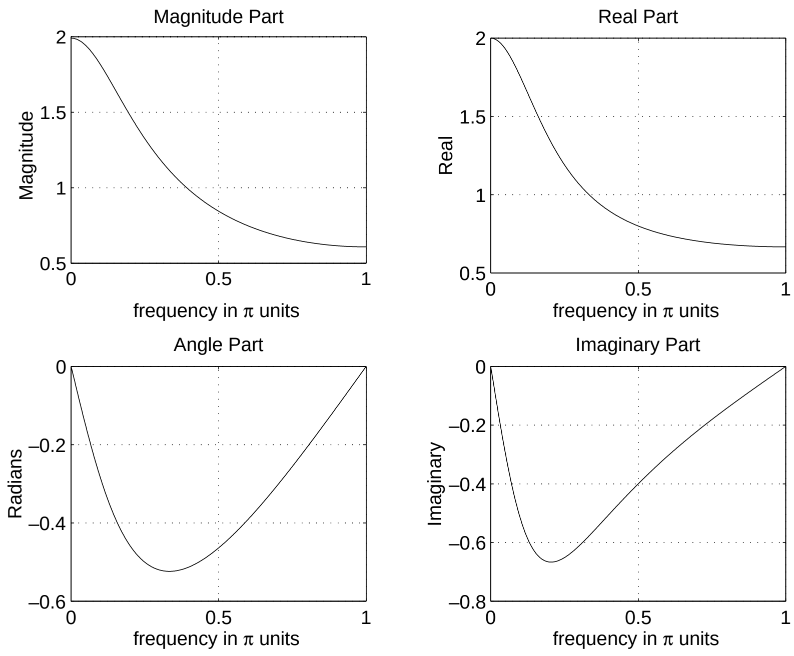


FIGURE 3.1 *Plots in Example 3.3*

The resulting plots are shown in Figure 3.1. Note that we divided the `w` array by `pi` before plotting so that the frequency axes are in the units of π and therefore easier to read. *This practice is strongly recommended.* □

If $x(n)$ is of finite duration, then MATLAB can be used to compute $X(e^{j\omega})$ numerically at any frequency ω. The approach is to implement (3.1) directly. If, in addition, we evaluate $X(e^{j\omega})$ at equispaced frequencies between $[0, \pi]$, then (3.1) can be implemented as a *matrix-vector multiplication* operation. To understand this, let us assume that the sequence $x(n)$ has N samples between $n_1 \le n \le n_N$ (i.e., not necessarily between $[0, N-1]$) and that we want to evaluate $X(e^{j\omega})$ at

$$\omega_k \overset{\triangle}{=} \frac{\pi}{M}k, \quad k = 0, 1, \ldots, M$$

which are $(M+1)$ equispaced frequencies between $[0, \pi]$. Then (3.1) can be written as

$$X(e^{j\omega_k}) = \sum_{\ell=1}^{N} e^{-j(\pi/M)kn_\ell} x(n_\ell), \quad k = 0, 1, \ldots, M$$

When $\{x(n_\ell)\}$ and $\{X(e^{j\omega_k})\}$ are arranged as *column* vectors $\mathbf{x}$ and $\mathbf{X}$, respectively, we have

$$\mathbf{X} = \mathbf{W}\mathbf{x} \tag{3.3}$$

where $\mathbf{W}$ is an $(M+1) \times N$ matrix given by

$$\mathbf{W} \triangleq \left\{e^{-j(\pi/M)kn_\ell};\ n_1 \le n \le n_N, \quad k = 0, 1, \dots, M\right\}$$

In addition, if we arrange $\{k\}$ and $\{n_\ell\}$ as *row* vectors $\mathbf{k}$ and $\mathbf{n}$ respectively, then

$$\mathbf{W} = \left[\exp\left(-j\frac{\pi}{M}\mathbf{k}^T\mathbf{n}\right)\right]$$

In MATLAB we represent sequences and indices as row vectors; therefore taking the transpose of (3.3), we obtain

$$\mathbf{X}^T = \mathbf{x}^T\left[\exp\left(-j\frac{\pi}{M}\mathbf{n}^T\mathbf{k}\right)\right] \tag{3.4}$$

Note that $\mathbf{n}^T\mathbf{k}$ is an $N \times (M+1)$ matrix. Now (3.4) can be implemented in MATLAB as follows.

```
>> k = [0:M]; n = [n1:n2];
>> X = x * (exp(-j*pi/M)) .^ (n'*k);
```

□ **EXAMPLE 3.4** Numerically compute the discrete-time Fourier transform of the sequence $x(n)$ given in Example 3.2 at 501 equispaced frequencies between $[0, \pi]$.

Solution MATLAB script:

```
>> n = -1:3; x = 1:5;  k = 0:500; w = (pi/500)*k;
>> X = x * (exp(-j*pi/500)) .^ (n'*k);
>> magX = abs(X); angX = angle(X);
>> realX = real(X); imagX = imag(X);
>> subplot(2,2,1); plot(k/500,magX);grid
>> xlabel('frequency in pi units'); title('Magnitude Part')
>> subplot(2,2,3); plot(k/500,angX/pi);grid
>> xlabel('frequency in pi units'); title('Angle Part')
>> subplot(2,2,2); plot(k/500,realX);grid
>> xlabel('frequency in pi units'); title('Real Part')
>> subplot(2,2,4); plot(k/500,imagX);grid
>> xlabel('frequency in pi units'); title('Imaginary Part')
```

The frequency-domain plots are shown in Figure 3.2. Note that the angle plot is depicted as a discontinuous function between $-\pi$ and π. This is because the `angle` function in MATLAB computes the principal angle. □

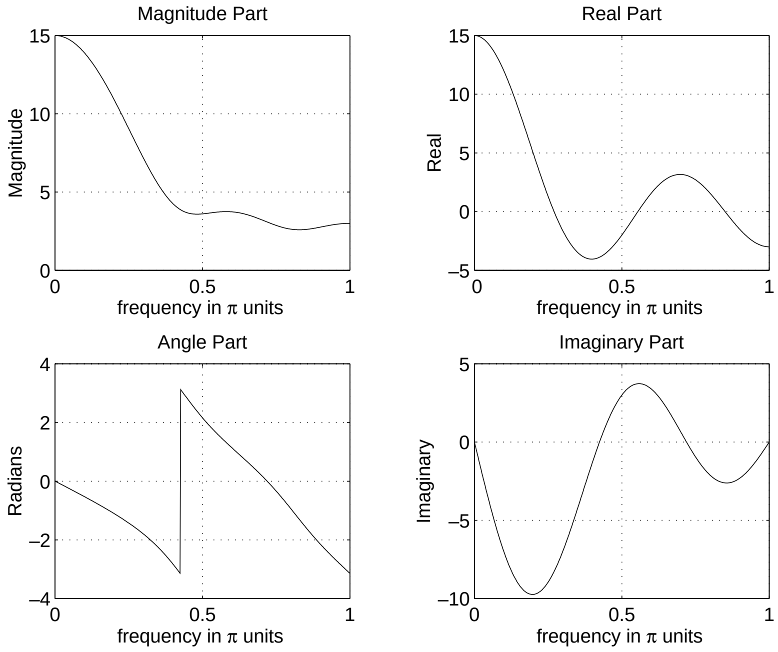


FIGURE 3.2 *Plots in Example 3.4*

The procedure of Example 3.4 can be compiled into a MATLAB function, say a `dtft` function, for ease of implementation. This is explored in Problem P3.1. This numerical computation is based on definition (3.1). It is not the most elegant way of numerically computing the discrete-time Fourier transform of a finite-duration sequence. In Chapter 5 we will discuss in detail the topic of a computable transform called the discrete Fourier transform (DFT) and its efficient computation called the fast Fourier transform (FFT). Also there is an alternate approach based on the z-transform using the MATLAB function `freqz`, which we will discuss in Chapter 4. In this chapter we will continue to use the approaches discussed so far for calculation as well as for investigation purposes.

In the next two examples we investigate the periodicity and symmetry properties using complex-valued and real-valued sequences.

□ **EXAMPLE 3.5** Let $x(n) = (0.9\exp(j\pi/3))^n\,,\; 0 \le n \le 10$. Determine $X(e^{j\omega})$ and investigate its periodicity.

Solution Since $x(n)$ is complex-valued, $X(e^{j\omega})$ satisfies only the periodicity property. Therefore it is uniquely defined over one period of 2π. However, we will evaluate and plot it at 401 frequencies over two periods between $[-2\pi, 2\pi]$ to observe its periodicity.

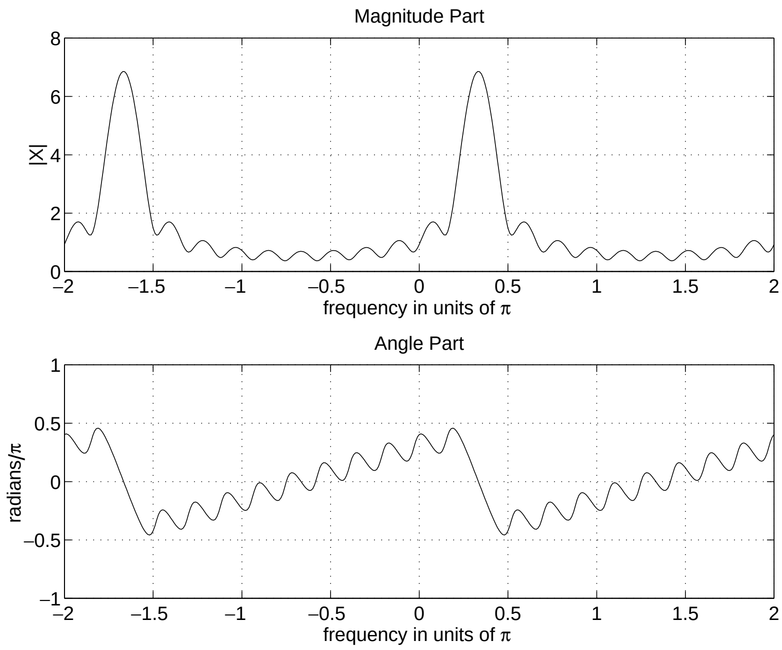


FIGURE 3.3 *Plots in Example 3.5*

MATLAB script:

```
>> n = 0:10; x = (0.9*exp(j*pi/3)).^n;
>> k = -200:200; w = (pi/100)*k;
>> X = x * (exp(-j*pi/100)) .^ (n'*k);
>> magX = abs(X); angX =angle(X);
>> subplot(2,1,1); plot(w/pi,magX);grid
>> xlabel('frequency in units of pi'); ylabel('|X|')
>> title('Magnitude Part')
>> subplot(2,1,2); plot(w/pi,angX/pi);grid
>> xlabel('frequency in units of pi'); ylabel('radians/pi')
>> title('Angle Part')
```

From the plots in Figure 3.3 we observe that $X(e^{j\omega})$ is periodic in ω but is not conjugate-symmetric. □

□ **EXAMPLE 3.6** Let $x(n) = (0.9)^n$, $-10 \le n \le 10$. Investigate the conjugate-symmetry property of its discrete-time Fourier transform.

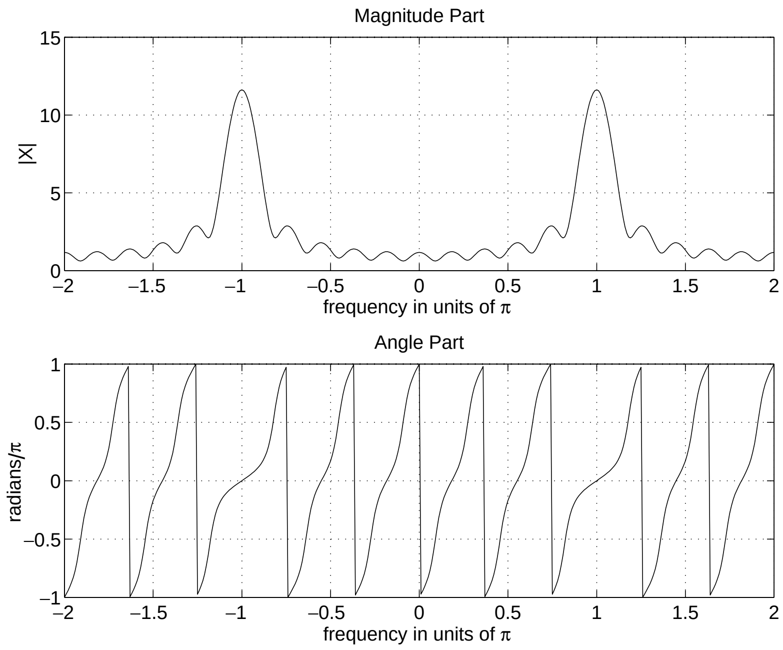


FIGURE 3.4 *Plots in Example 3.6*

Solution Once again we will compute and plot $X(e^{j\omega})$ over two periods to study its symmetry property.

MATLAB script:

```
>> n = -5:5; x = (-0.9).^n;
>> k = -200:200; w = (pi/100)*k;  X = x * (exp(-j*pi/100)) .^ (n'*k);
>> magX = abs(X); angX =angle(X);
>> subplot(2,1,1); plot(w/pi,magX);grid;  axis([-2,2,0,15])
>> xlabel('frequency in units of pi'); ylabel('|X|')
>> title('Magnitude Part')
>> subplot(2,1,2); plot(w/pi,angX/pi);grid;  axis([-2,2,-1,1])
>> xlabel('frequency in units of pi'); ylabel('radians/pi')
>> title('Angle Part')
```

From the plots in Figure 3.4 we observe that $X(e^{j\omega})$ is not only periodic in ω but is also conjugate-symmetric. Therefore for real sequences we will plot their Fourier transform magnitude and angle graphs from 0 to π. □

3.1.3 SOME COMMON DTFT PAIRS

The discrete-time Fourier transforms of the basic sequences discussed in Chapter 2 are very useful. The discrete-time Fourier transforms of some

TABLE 3.1 *Some common DTFT pairs*

Signal Type	*Sequence* $x(n)$	DTFT $X(e^{j\omega})$, $-\pi \le \omega \le \pi$
Unit impulse	$\delta(n)$	1
Constant	1	$2\pi\delta(\omega)$
Unit step	$u(n)$	$\dfrac{1}{1-e^{-j\omega}} + \pi\delta(\omega)$
Causal exponential	$\alpha^n u(n)$	$\dfrac{1}{1-\alpha e^{-j\omega}}$
Complex exponential	$e^{j\omega_0 n}$	$2\pi\delta(\omega-\omega_0)$
Cosine	$\cos(\omega_0 n)$	$\pi[\delta(\omega-\omega_0)+\delta(\omega+\omega_0)]$
Sine	$\sin(\omega_0 n)$	$j\pi[\delta(\omega+\omega_0)-\delta(\omega-\omega_0)]$
Double exponential	$\alpha^{\lvert n\rvert} u(n)$	$\dfrac{1-\alpha^2}{1-2\alpha\cos(\omega)+\alpha^2}$

Note: Since $X(e^{j\omega})$ is periodic with period 2π, expressions over only the primary period of $-\pi \le \omega \le \pi$ are given.

of these sequences can be easily obtained using the basic definitions (3.1) and (3.2). These transform pairs and those of few other pairs are given in Table 3.1. Note that, even if sequences like unit step $u(n)$ are not absolutely summable, their discrete-time Fourier transforms exist in the limiting sense if we allow impulses in the Fourier transform. Such sequences are said to have finite power, that is, $\sum_n |x(n)|^2 < \infty$. Using this table and the properties of the Fourier transform (discussed in Section 3.2), it is possible to obtain discrete-time Fourier transform of many more sequences.

3.2 THE PROPERTIES OF THE DTFT

In the previous section, we discussed two important properties that we needed for plotting purposes. We now discuss the remaining useful properties, which are given below without proof. Let $X(e^{j\omega})$ be the discrete-time Fourier transform of $x(n)$.

1. **Linearity:** The discrete-time Fourier transform is a linear transformation; that is,

$$\mathcal{F}[\alpha x_1(n) + \beta x_2(n)] = \alpha\mathcal{F}[x_1(n)] + \beta\mathcal{F}[x_2(n)] \tag{3.5}$$

for every α, β, $x_1(n)$, and $x_2(n)$.

2. **Time shifting:** A shift in the time domain corresponds to the phase shifting.

$$\mathcal{F}\left[x(n-k)\right] = X(e^{j\omega})e^{-j\omega k} \tag{3.6}$$

3. **Frequency shifting:** Multiplication by a complex exponential corresponds to a shift in the frequency domain.

$$\mathcal{F}\left[x(n)e^{j\omega_0 n}\right] = X(e^{j(\omega-\omega_0)}) \tag{3.7}$$

4. **Conjugation:** Conjugation in the time domain corresponds to the folding and conjugation in the frequency domain.

$$\mathcal{F}\left[x^*(n)\right] = X^*(e^{-j\omega}) \tag{3.8}$$

5. **Folding:** Folding in the time domain corresponds to the folding in the frequency domain.

$$\mathcal{F}\left[x(-n)\right] = X(e^{-j\omega}) \tag{3.9}$$

6. **Symmetries in real sequences:** We have already studied the conjugate symmetry of real sequences. These real sequences can be decomposed into their even and odd parts, as discussed in Chapter 2.

$$x(n) = x_e(n) + x_o(n)$$

Then

$$\begin{aligned} \mathcal{F}\left[x_e(n)\right] &= \text{Re}\left[X(e^{j\omega})\right] \\ \mathcal{F}\left[x_o(n)\right] &= j\,\text{Im}\left[X(e^{j\omega})\right] \end{aligned} \tag{3.10}$$

Implication: If the sequence $x(n)$ is real and even, then $X(e^{j\omega})$ is also real and even. Hence only one plot over $[0, \pi]$ is necessary for its complete representation.

A similar property for complex-valued sequences is explored in Problem P3.7.

7. **Convolution:** This is one of the most useful properties that makes system analysis convenient in the frequency domain.

$$\mathcal{F}\left[x_1(n) * x_2(n)\right] = \mathcal{F}\left[x_1(n)\right]\mathcal{F}\left[x_2(n)\right] = X_1(e^{j\omega})X_2(e^{j\omega}) \tag{3.11}$$

8. **Multiplication:** This is a dual of the convolution property.

$$\mathcal{F}\left[x_1(n)\cdot x_2(n)\right] = \mathcal{F}\left[x_1(n)\right] \circledast \mathcal{F}[x_2(n)] \triangleq \frac{1}{2\pi}\int_{-\pi}^{\pi} X_1(e^{j\theta})X_2(e^{j(\omega-\theta)})d\theta \tag{3.12}$$

This convolution-like operation is called a *periodic convolution* and hence denoted by $\circledast$. It is discussed (in its discrete form) in Chapter 5.

9. **Energy:** The energy of the sequence $x(n)$ can be written as

$$\mathcal{E}_x = \sum_{-\infty}^{\infty} |x(n)|^2 = \frac{1}{2\pi}\int_{-\pi}^{\pi} |X(e^{j\omega})|^2 d\omega \tag{3.13}$$

$$= \int_{0}^{\pi} \frac{|X(e^{j\omega})|^2}{\pi} d\omega \quad \text{(for real sequences using even symmetry)}$$

This is also known as Parseval's theorem. From (3.13) the *energy density spectrum* of $x(n)$ is defined as

$$\Phi_x(\omega) \triangleq \frac{|X(e^{j\omega})|^2}{\pi} \tag{3.14}$$

Then the energy of $x(n)$ in the $[\omega_1, \omega_2]$ band is given by

$$\int_{\omega_1}^{\omega_2} \Phi_x(\omega)d\omega, \quad 0 \le \omega_1 < \omega_2 \le \pi$$

In the next several examples we will verify some of these properties using finite-duration sequences. We will follow our numerical procedure to compute discrete-time Fourier transforms in each case. Although this does not analytically prove the validity of each property, it provides us with an experimental tool in practice.

□ **EXAMPLE 3.7** In this example we will verify the linearity property (3.5) using real-valued finite-duration sequences. Let $x_1(n)$ and $x_2(n)$ be two random sequences uniformly distributed between $[0, 1]$ over $0 \le n \le 10$. Then we can use our numerical discrete-time Fourier transform procedure as follows.

MATLAB script:

```
>> x1 = rand(1,11); x2 = rand(1,11); n = 0:10;
>> alpha = 2; beta = 3;  k = 0:500; w = (pi/500)*k;
>> X1 = x1 * (exp(-j*pi/500)).^(n'*k);  % DTFT of x1
>> X2 = x2 * (exp(-j*pi/500)).^(n'*k);  % DTFT of x2
>> x = alpha*x1 + beta*x2;              % Linear combination of x1 & x2
>> X = x * (exp(-j*pi/500)).^(n'*k);    % DTFT of x
>> % verification
>> X_check = alpha*X1 + beta*X2;        % Linear Combination of X1 & X2
>> error = max(abs(X-X_check))          % Difference
error =
  7.1054e-015
```

Since the maximum absolute error between the two Fourier transform arrays is less than 10^{-14}, the two arrays are identical within the limited numerical precision of MATLAB. □

□ **EXAMPLE 3.8** Let $x(n)$ be a random sequence uniformly distributed between $[0, 1]$ over $0 \leq n \leq 10$ and let $y(n) = x(n-2)$. Then we can verify the sample shift property (3.6) as follows.

```
>> x = rand(1,11); n = 0:10;
>> k = 0:500; w = (pi/500)*k;
>> X = x * (exp(-j*pi/500)).^(n'*k);    % DTFT of x
>> % signal shifted by two samples
>> y = x; m = n+2;
>> Y = y * (exp(-j*pi/500)).^(m'*k);    % DTFT of y
>> % verification
>> Y_check = (exp(-j*2).^w).*X;         % multiplication by exp(-j2w)
>> error = max(abs(Y-Y_check))          % Difference
error =
  5.7737e-015
```
□

□ **EXAMPLE 3.9** To verify the frequency shift property (3.7), we will use the graphical approach. Let

$$x(n) = \cos(\pi n/2), \quad 0 \leq n \leq 100 \qquad \text{and} \qquad y(n) = e^{j\pi n/4}x(n)$$

Then using MATLAB,

```
>> n = 0:100; x = cos(pi*n/2);
>> k = -100:100; w = (pi/100)*k;        % frequency between -pi and +pi
>> X = x * (exp(-j*pi/100)).^(n'*k);    % DTFT of x
%
>> y = exp(j*pi*n/4).*x;                % signal multiplied by exp(j*pi*n/4)
>> Y = y * (exp(-j*pi/100)).^(n'*k);    % DTFT of y
% Graphical verification
>> subplot(2,2,1); plot(w/pi,abs(X)); grid; axis([-1,1,0,60])
>> xlabel('frequency in pi units'); ylabel('|X|')
>> title('Magnitude of X')
>> subplot(2,2,2); plot(w/pi,angle(X)/pi); grid; axis([-1,1,-1,1])
>> xlabel('frequency in pi units'); ylabel('radiands/pi')
>> title('Angle of X')
>> subplot(2,2,3); plot(w/pi,abs(Y)); grid; axis([-1,1,0,60])
```

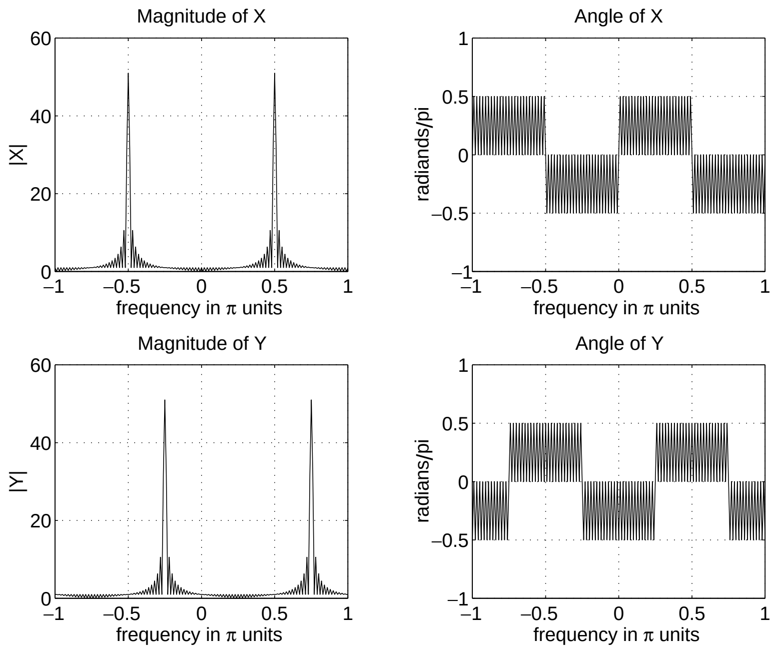


FIGURE 3.5 *Plots in Example 3.9*

```
>> xlabel('frequency in pi units'); ylabel('|Y|')
>> title('Magnitude of Y')
>> subplot(2,2,4); plot(w/pi,angle(Y)/pi); grid; axis([-1,1,-1,1])
>> xlabel('frequency in pi units'); ylabel('radians/pi')
>> title('Angle of Y')
```

From the plots in Figure 3.5, we observe that $X(e^{j\omega})$ is indeed shifted by $\pi/4$ in both magnitude and angle. □

□ **EXAMPLE 3.10** To verify the conjugation property (3.8), let $x(n)$ be a complex-valued random sequence over $-5 \le n \le 10$ with real and imaginary parts uniformly distributed between $[0,1]$. The MATLAB verification is as follows.

```
>> n = -5:10; x = rand(1,length(n)) + j*rand(1,length(n));
>> k = -100:100; w = (pi/100)*k;        % frequency between -pi and +pi
>> X = x * (exp(-j*pi/100)).^(n'*k);    % DTFT of x
% conjugation property
>> y = conj(x);                         % signal conjugation
>> Y = y * (exp(-j*pi/100)).^(n'*k);    % DTFT of y
% verification
```

```
>> Y_check = conj(fliplr(X));          % conj(X(-w))
>> error = max(abs(Y-Y_check))         % Difference
error =
     0
```

□

□ **EXAMPLE 3.11** To verify the folding property (3.9), let $x(n)$ be a random sequence over $-5 \le n \le 10$ uniformly distributed between $[0, 1]$. The MATLAB verification is as follows.

```
>> n = -5:10; x = rand(1,length(n));
>> k = -100:100; w = (pi/100)*k;       % frequency between -pi and +pi
>> X = x * (exp(-j*pi/100)).^(n'*k);   % DTFT of x
% folding property
>> y = fliplr(x); m = -fliplr(n);      % signal folding
>> Y = y * (exp(-j*pi/100)).^(m'*k);   % DTFT of y
% verification
>> Y_check = fliplr(X);                % X(-w)
>> error = max(abs(Y-Y_check))         % Difference
error =
     0
```

□

□ **EXAMPLE 3.12** In this problem we verify the symmetry property (3.10) of real signals. Let

$$x(n) = \sin(\pi n/2), \quad -5 \le n \le 10$$

Then using the **evenodd** function developed in Chapter 2, we can compute the even and odd parts of $x(n)$ and then evaluate their discrete-time Fourier transforms. We will provide the numerical as well as graphical verification.

MATLAB script:

```
>> n = -5:10; x = sin(pi*n/2);
>> k = -100:100; w = (pi/100)*k;       % frequency between -pi and +pi
>> X = x * (exp(-j*pi/100)).^(n'*k);   % DTFT of x
% signal decomposition
>> [xe,xo,m] = evenodd(x,n);           % even and odd parts
>> XE = xe * (exp(-j*pi/100)).^(m'*k); % DTFT of xe
>> XO = xo * (exp(-j*pi/100)).^(m'*k); % DTFT of xo
% verification
>> XR = real(X);                       % real part of X
>> error1 = max(abs(XE-XR))            % Difference
error1 =
  1.8974e-019
>> XI = imag(X);                       % imag part of X
```

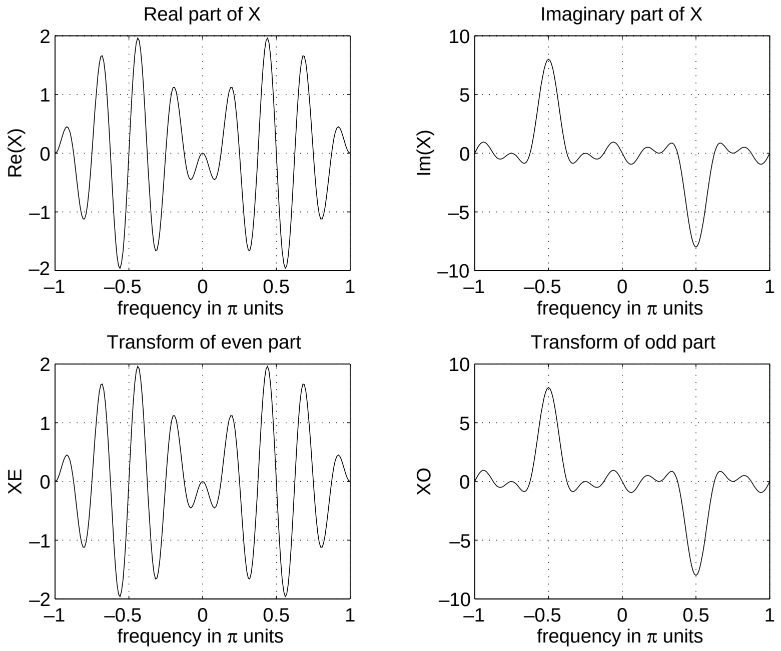


FIGURE 3.6 *Plots in Example 3.12*

```
>> error2 = max(abs(XO-j*XI))          % Difference
error2 =
  1.8033e-019
% graphical verification
>> subplot(2,2,1); plot(w/pi,XR); grid; axis([-1,1,-2,2])
>> xlabel('frequency in pi units'); ylabel('Re(X)');
>> title('Real part of X')
>> subplot(2,2,2); plot(w/pi,XI); grid; axis([-1,1,-10,10])
>> xlabel('frequency in pi units'); ylabel('Im(X)');
>> title('Imaginary part of X')
>> subplot(2,2,3); plot(w/pi,real(XE)); grid; axis([-1,1,-2,2])
>> xlabel('frequency in pi units'); ylabel('XE');
>> title('Transform of even part')
>> subplot(2,2,4); plot(w/pi,imag(XO)); grid; axis([-1,1,-10,10])
>> xlabel('frequency in pi units'); ylabel('XO');
>> title('Transform of odd part')
```

From the plots in Figure 3.6 we observe that the real part of $X(e^{j\omega})$ [or the imaginary part of $X(e^{j\omega})$] is equal to the discrete-time Fourier transform of $x_e(n)$ [or $x_o(n)$]. □

3.3 THE FREQUENCY DOMAIN REPRESENTATION OF LTI SYSTEMS

We earlier stated that the Fourier transform representation is the most useful signal representation for LTI systems. It is due to the following result.

3.3.1 RESPONSE TO A COMPLEX EXPONENTIAL $e^{j\omega_0 n}$

Let $x(n) = e^{j\omega_0 n}$ be the input to an LTI system represented by the impulse response $h(n)$.

$$e^{j\omega_0 n} \longrightarrow \boxed{h(n)} \longrightarrow h(n) * e^{j\omega_0 n}$$

Then

$$\begin{aligned} y(n) = h(n) * e^{j\omega_0 n} &= \sum_{-\infty}^{\infty} h(k) e^{j\omega_0 (n-k)} \\ &= \left[\sum_{-\infty}^{\infty} h(k) e^{-j\omega_0 k}\right] e^{j\omega_0 n} \qquad \textbf{(3.15)} \\ &= \left[\mathcal{F}[h(n)]|_{\omega=\omega_0}\right] e^{j\omega_0 n} \end{aligned}$$

■ **DEFINITION 1** [Frequency Response] The discrete-time Fourier transform of an impulse response is called the *frequency response* (or *transfer function*) of an LTI system and is denoted by

$$H(e^{j\omega n}) \triangleq \sum_{-\infty}^{\infty} h(n) e^{-j\omega n} \qquad \textbf{(3.16)}$$

Then from (3.15) we can represent the system by

$$x(n) = e^{j\omega_0 n} \longrightarrow \boxed{H(e^{j\omega})} \longrightarrow y(n) = H(e^{j\omega_0}) \times e^{j\omega_0 n} \qquad \textbf{(3.17)}$$

Hence the output sequence is the input exponential sequence *modified* by the response of the system at frequency ω_0. This justifies the definition of $H(e^{j\omega})$ as a frequency response because it is what the complex exponential is multiplied by to obtain the output $y(n)$. This powerful result can be extended to a linear combination of complex exponentials using the linearity of LTI systems.

$$\sum_k A_k e^{j\omega_k n} \longrightarrow \boxed{h(n)} \longrightarrow \sum_k A_k H(e^{j\omega_k})\, e^{j\omega_k n}$$

In general, the frequency response $H(e^{j\omega})$ is a complex function of ω. The magnitude $|H(e^{j\omega})|$ of $H(e^{j\omega})$ is called the *magnitude* (*or gain*) *response* function, and the angle $\angle H(e^{j\omega})$ is called the *phase response* function as we shall see below.

3.3.2 RESPONSE TO SINUSOIDAL SEQUENCES

Let $x(n) = A\cos(\omega_0 n + \theta_0)$ be an input to an LTI system $h(n)$. Then from (3.17) we can show that the response $y(n)$ is another sinusoid of the same frequency ω_0, with amplitude *gained* by $|H(e^{j\omega_0})|$ and phase *shifted* by $\angle H(e^{j\omega_0})$, that is,

$$y(n) = A|H(e^{j\omega_0})|\cos(\omega_0 n + \theta_0 + \angle H(e^{j\omega_0})) \tag{3.18}$$

This response is called the *steady-state response*, denoted by $y_{ss}(n)$. It can be extended to a linear combination of sinusoidal sequences.

$$\sum_k A_k \cos(\omega_k n + \theta_k) \longrightarrow \boxed{H(e^{j\omega})} \longrightarrow \textstyle\sum_k A_k |H(e^{j\omega_k})| \cos(\omega_k n + \theta_k + \angle H(e^{j\omega_k}))$$

3.3.3 RESPONSE TO ARBITRARY SEQUENCES

Finally, (3.17) can be generalized to arbitrary *absolutely summable* sequences. Let $X(e^{j\omega}) = \mathcal{F}[x(n)]$ and $Y(e^{j\omega}) = \mathcal{F}[y(n)]$; then using the convolution property (3.11), we have

$$Y(e^{j\omega}) = H(e^{j\omega})\, X(e^{j\omega}) \tag{3.19}$$

Therefore an LTI system can be represented in the frequency domain by

$$X(e^{j\omega}) \longrightarrow \boxed{H(e^{j\omega})} \longrightarrow Y(e^{j\omega}) = H(e^{j\omega})\, X(e^{j\omega})$$

The output $y(n)$ is then computed from $Y(e^{j\omega})$ using the inverse discrete-time Fourier transform (3.2). This requires an integral operation, which is not a convenient operation in MATLAB. As we shall see in Chapter 4, there is an alternate approach to the computation of output to arbitrary inputs using the z-transform and partial fraction expansion. In this chapter we will concentrate on computing the steady-state response.

□ **EXAMPLE 3.13** Determine the frequency response $H(e^{j\omega})$ of a system characterized by $h(n) = (0.9)^n u(n)$. Plot the magnitude and the phase responses.

Solution Using (3.16),

$$H(e^{j\omega}) = \sum_{-\infty}^{\infty} h(n)e^{-j\omega n} = \sum_{0}^{\infty}(0.9)^n e^{-j\omega n}$$

$$= \sum_{0}^{\infty}(0.9e^{-j\omega})^n = \frac{1}{1-0.9e^{-j\omega}}$$

Hence

$$|H(e^{j\omega})| = \sqrt{\frac{1}{(1-0.9\cos\omega)^2 + (0.9\sin\omega)^2}} = \frac{1}{\sqrt{1.81 - 1.8\cos\omega}}$$

and

$$\angle H(e^{j\omega}) = -\arctan\left[\frac{0.9\sin\omega}{1-0.9\cos\omega}\right]$$

To plot these responses, we can either implement the $|H(e^{j\omega})|$ and $\angle H(e^{j\omega})$ functions or the frequency response $H(e^{j\omega})$ and then compute its magnitude and phase. The latter approach is more useful from a practical viewpoint [as shown in (3.18)].

```
>> w = [0:1:500]*pi/500;  % [0, pi] axis divided into 501 points.
>> H = exp(j*w) ./ (exp(j*w) - 0.9*ones(1,501));
>> magH = abs(H); angH = angle(H);
>> subplot(2,1,1); plot(w/pi,magH); grid;
>> xlabel('frequency in pi units'); ylabel('|H|');
>> title('Magnitude Response');
>> subplot(2,1,2); plot(w/pi,angH/pi); grid
>> xlabel('frequency in pi units'); ylabel('Phase in pi Radians');
>> title('Phase Response');
```

The plots are shown in Figure 3.7. □

□ **EXAMPLE 3.14** Let an input to the system in Example 3.13 be $0.1u(n)$. Determine the steady-state response $y_{ss}(n)$.

Solution Since the input is not absolutely summable, the discrete-time Fourier transform is not particularly useful in computing the complete response. However, it can be used to compute the steady-state response. In the steady state (i.e., $n \to \infty$), the input is a constant sequence (or a sinusoid with $\omega_0 = \theta_0 = 0$). Then the output is

$$y_{ss}(n) = 0.1 \times H(e^{j0}) = 0.1 \times 10 = 1$$

where the gain of the system at $\omega = 0$ (also called the DC gain) is $H(e^{j0}) = 10$, which is obtained from Figure 3.7. □

3.3.4 FREQUENCY RESPONSE FUNCTION FROM DIFFERENCE EQUATIONS

When an LTI system is represented by the difference equation

$$y(n) + \sum_{\ell=1}^{N} a_\ell y(n-\ell) = \sum_{m=0}^{M} b_m x(n-m) \tag{3.20}$$

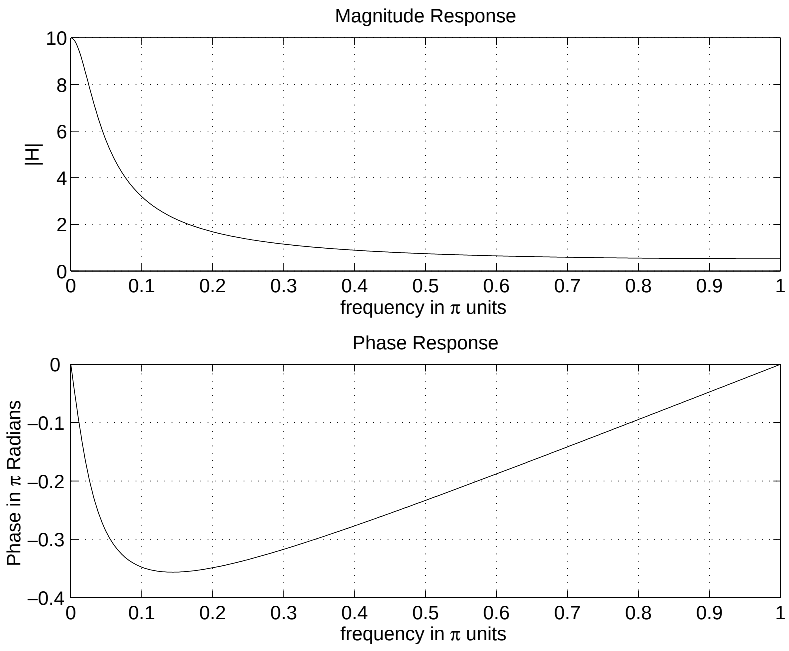


FIGURE 3.7 *Frequency response plots in Example 3.13*

then to evaluate its frequency response from (3.16), we would need the impulse response $h(n)$. However, using (3.17), we can easily obtain $H(e^{j\omega})$. We know that when $x(n) = e^{j\omega n}$, then $y(n)$ must be $H(e^{j\omega})e^{j\omega n}$. Substituting in (3.20), we have

$$H(e^{j\omega})e^{j\omega n} + \sum_{\ell=1}^{N} a_\ell H(e^{j\omega})e^{j\omega(n-\ell)} = \sum_{m=0}^{M} b_m \; e^{j\omega(n-m)}$$

or

$$H(e^{j\omega}) = \frac{\sum_{m=0}^{M} b_m \; e^{-j\omega m}}{1 + \sum_{\ell=1}^{N} a_\ell \; e^{-j\omega \ell}} \tag{3.21}$$

after canceling the common factor $e^{j\omega n}$ term and rearranging. This equation can easily be implemented in MATLAB, given the difference equation parameters.

□ **EXAMPLE 3.15** An LTI system is specified by the difference equation

$$y(n) = 0.8y(n-1) + x(n)$$

a. Determine $H(e^{j\omega})$.
b. Calculate and plot the steady-state response $y_{ss}(n)$ to

$$x(n) = \cos(0.05\pi n)u(n)$$

Solution Rewrite the difference equation as $y(n) - 0.8y(n-1) = x(n)$.

a. Using (3.21), we obtain

$$H(e^{j\omega}) = \frac{1}{1 - 0.8e^{-j\omega}} \tag{3.22}$$

b. In the steady state the input is $x(n) = \cos(0.05\pi n)$ with frequency $\omega_0 = 0.05\pi$ and $\theta_0 = 0^\circ$. The response of the system is

$$H(e^{j0.05\pi}) = \frac{1}{1 - 0.8e^{-j0.05\pi}} = 4.0928e^{-j0.5377}$$

Therefore

$$y_{ss}(n) = 4.0928\cos(0.05\pi n - 0.5377) = 4.0928\cos\left[0.05\pi(n - 3.42)\right]$$

This means that at the output the sinusoid is scaled by 4.0928 and shifted by 3.42 samples. This can be verified using MATLAB.

```
>> subplot(1,1,1)
>> b = 1; a = [1,-0.8];
>> n=[0:100];x = cos(0.05*pi*n);
>> y = filter(b,a,x);
>> subplot(2,1,1); stem(n,x);
>> xlabel('n'); ylabel('x(n)'); title('Input sequence')
>> subplot(2,1,2); stem(n,y);
>> xlabel('n'); ylabel('y(n)'); title('Output sequence')
```

From the plots in Figure 3.8, we note that the amplitude of $y_{ss}(n)$ is approximately 4. To determine the shift in the output sinusoid, we can compare zero crossings of the input and the output. This is shown in Figure 3.8, from which the shift is approximately 3.4 samples. □

In Example 3.15 the system was characterized by a 1st-order difference equation. It is fairly straightforward to implement (3.22) in MATLAB as we did in Example 3.13. In practice the difference equations are of large order and hence we need a compact procedure to implement the general expression (3.21). This can be done using a simple matrix-vector multiplication. If we evaluate $H(e^{j\omega})$ at $k = 0, 1, \ldots, K$ equispaced frequencies over $[0, \pi]$, then

$$H(e^{j\omega_k}) = \frac{\sum_{m=0}^{M} b_m \, e^{-j\omega_k m}}{1 + \sum_{\ell=1}^{N} a_\ell \, e^{-j\omega_k \ell}}, \quad k = 0, 1, \ldots, K \tag{3.23}$$

If we let $\{b_m\}$, $\{a_\ell\}$ (with $a_0 = 1$), $\{m = 0, \ldots, M\}$, $\{\ell = 0, \ldots, N\}$, and $\{\omega_k\}$ be arrays (or row vectors), then the numerator and the denominator of (3.23) become

$$\underline{b}\exp(-j\underline{m}^T\underline{\omega}); \quad \underline{a}\exp(-j\underline{\ell}^T\underline{\omega})$$

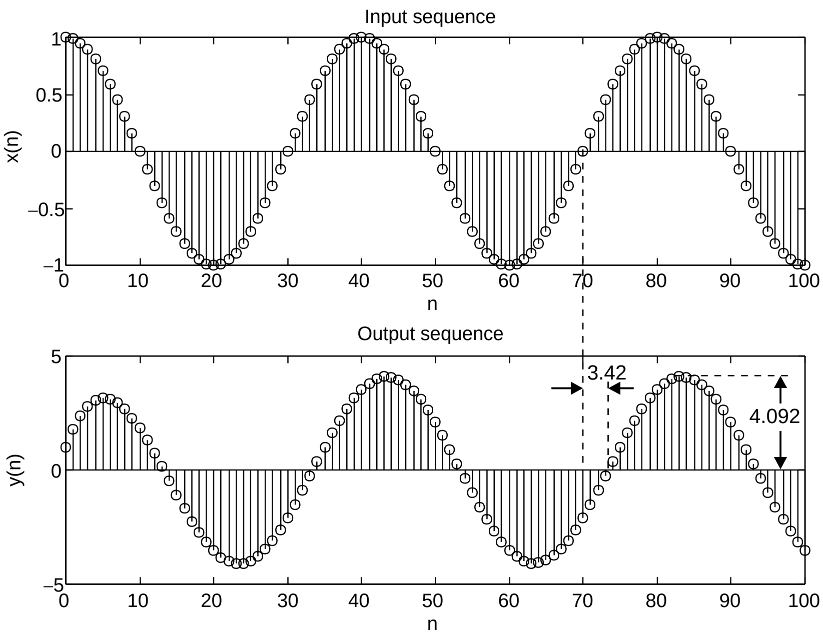


FIGURE 3.8 *Plots in Example 3.15*

respectively. Now the array $H(e^{j\omega_k})$ in (3.23) can be computed using a `./` operation. This procedure can be implemented in a MATLAB function to determine the frequency response function, given $\{b_m\}$ and $\{a_\ell\}$ arrays. We will explore this in Example 3.16 and in Problem P3.16.

□ **EXAMPLE 3.16** A 3rd-order lowpass filter is described by the difference equation

$$\begin{aligned} y(n) &= 0.0181x(n) + 0.0543x(n-1) + 0.0543x(n-2) + 0.0181x(n-3) \\ &\quad +1.76y(n-1) - 1.1829y(n-2) + 0.2781y(n-3) \end{aligned}$$

Plot the magnitude and the phase response of this filter, and verify that it is a lowpass filter.

Solution We will implement this procedure in MATLAB and then plot the filter responses.

```
>> b = [0.0181,  0.0543, 0.0543,  0.0181]; % filter coefficient array b
>> a = [1.0000, -1.7600, 1.1829, -0.2781]; % filter coefficient array a
>> m = 0:length(b)-1; l = 0:length(a)-1;   % index arrays m and l
>> K = 500; k = 0:1:K;                     % index array k for frequencies
>> w = pi*k/K;                             % [0, pi] axis divided into 501 points.
>> num = b * exp(-j*m'*w);                 % Numerator calculations
>> den = a * exp(-j*l'*w);                 % Denominator calculations
>> H = num ./ den;                         % Frequency response
>> magH = abs(H); angH = angle(H);         % mag and phase responses
```

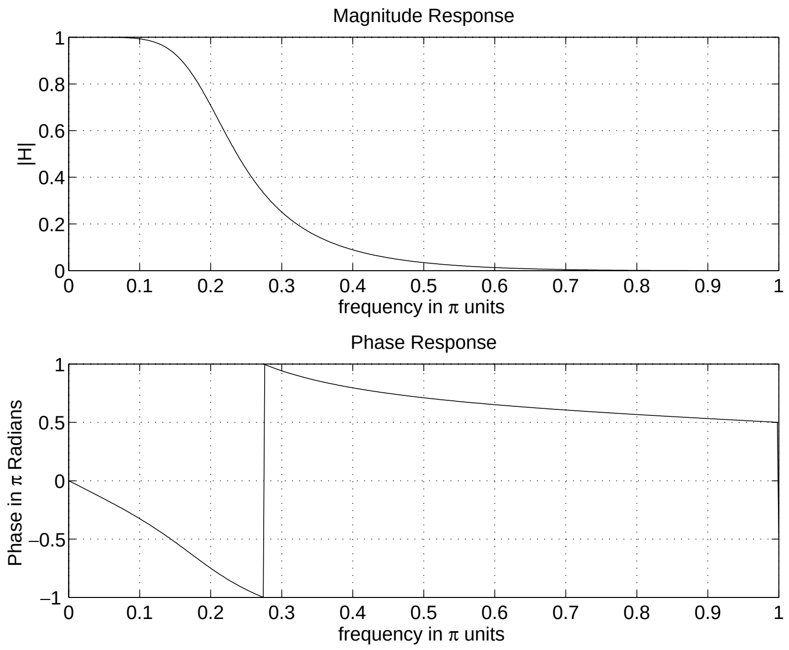


FIGURE 3.9 *Plots for Example 3.16*

```
>> subplot(2,1,1); plot(w/pi,magH); grid; axis([0,1,0,1])
>> xlabel('frequency in pi units'); ylabel('|H|');
>> title('Magnitude Response');
>> subplot(2,1,2); plot(w/pi,angH/pi); grid
>> xlabel('frequency in pi units'); ylabel('Phase in pi Radians');
>> title('Phase Response');
```

From the plots in Figure 3.9 we see that the filter is indeed a lowpass filter. □

3.4 SAMPLING AND RECONSTRUCTION OF ANALOG SIGNALS

In many applications—for example, in digital communications—real-world analog signals are converted into discrete signals using sampling and quantization operations (collectively called analog-to-digital conversion, or ADC). These discrete signals are processed by digital signal processors, and the processed signals are converted into analog signals using a reconstruction operation (called digital-to-analog conversion or

DAC). Using Fourier analysis, we can describe the sampling operation from the frequency-domain viewpoint, analyze its effects, and then address the reconstruction operation. We will also assume that the number of quantization levels is sufficiently large that the effect of quantization on discrete signals is negligible. We will study the effects of quantization in Chapter 10.

3.4.1 SAMPLING

Let $x_a(t)$ be an analog (absolutely integrable) signal. Its continuous-time Fourier transform (CTFT) is given by

$$X_a(j\Omega) \triangleq \int_{-\infty}^{\infty} x_a(t)e^{-j\Omega t}dt \tag{3.24}$$

where Ω is an analog frequency in radians/sec. The inverse continuous-time Fourier transform is given by

$$x_a(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X_a(j\Omega)e^{j\Omega t}d\Omega \tag{3.25}$$

We now sample $x_a(t)$ at *sampling interval* T_s seconds apart to obtain the discrete-time signal $x(n)$.

$$x(n) \triangleq x_a(\ nT_s)$$

Let $X(e^{j\omega})$ be the discrete-time Fourier transform of $x(n)$. Then it can be shown [23] that $X(e^{j\omega})$ is a countable sum of amplitude-scaled, frequency-scaled, and translated versions of the Fourier transform $X_a(j\Omega)$.

$$X(e^{j\omega}) = \frac{1}{T_s}\sum_{\ell=-\infty}^{\infty} X_a\left[j\left(\frac{\omega}{T_s} - \frac{2\pi}{T_s}\ell\right)\right] \tag{3.26}$$

This relation is known as the *aliasing formula*. The analog and digital frequencies are related through T_s

$$\omega = \Omega T_s \tag{3.27}$$

while the sampling frequency F_s is given by

$$F_s \triangleq \frac{1}{T_s}, \quad \text{sam/sec} \tag{3.28}$$

The graphical illustration of (3.26) is shown in Figure 3.10, from which we observe that, in general, the discrete signal is an *aliased version* of the corresponding analog signal because higher frequencies are aliased into lower frequencies if there is an overlap. However, it is possible to recover the Fourier transform $X_a(j\Omega)$ from $X(e^{j\omega})$ [or equivalently, the analog

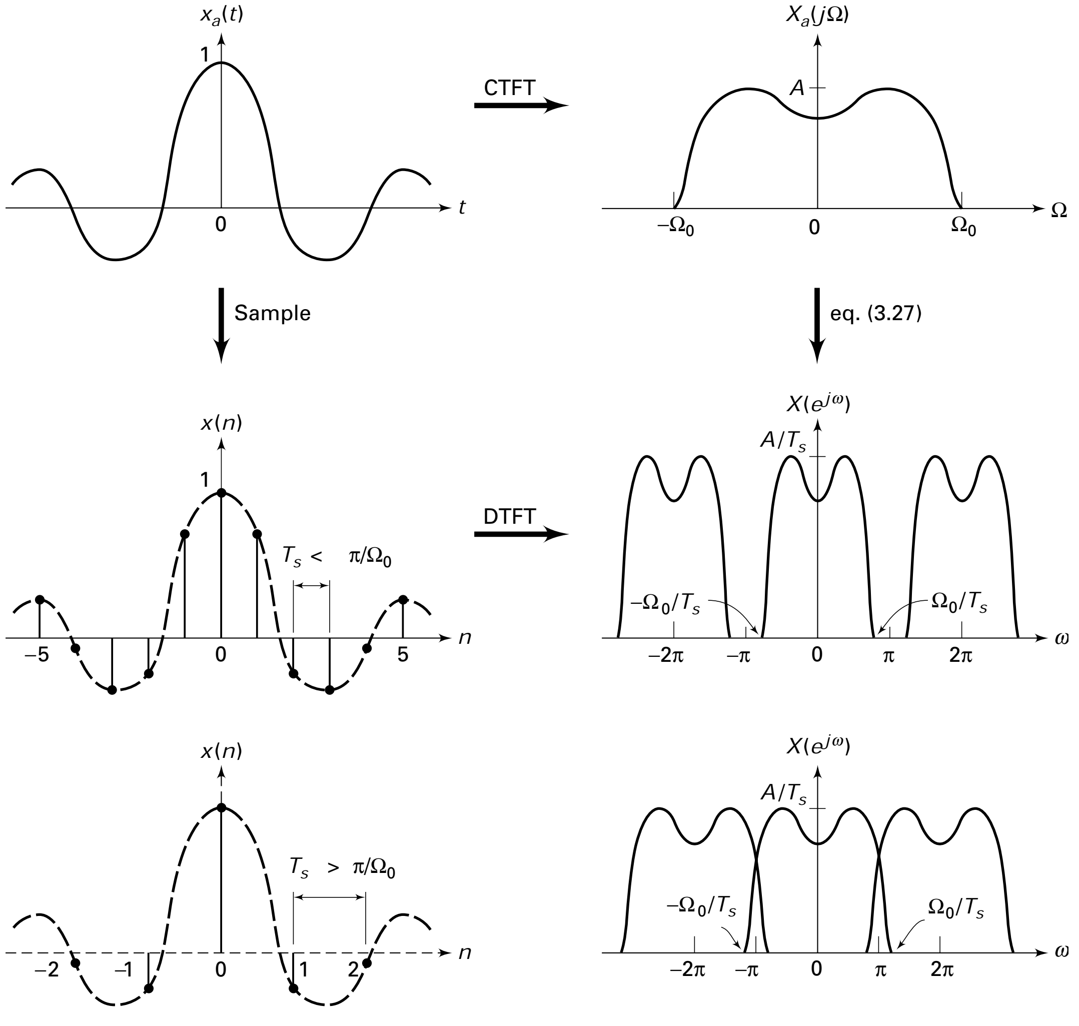


FIGURE 3.10 *Sampling operation in the time and frequency domains*

signal $x_a(t)$ from its samples $x(n)$] if the infinite "replicas" of $X_a(j\Omega)$ do not overlap with each other to form $X(e^{j\omega})$. This is true for band-limited analog signals.

■ **DEFINITION 2** [Band-limited Signal] A signal is band-limited if there exists a finite radian frequency Ω_0 such that $X_a(j\Omega)$ is zero for $|\Omega| > \Omega_0$. The frequency $F_{0=}\Omega_0/2\pi$ is called the signal bandwidth in Hz.

Referring to Figure 3.10, if $\pi > \Omega_0 T_s$—or equivalently, $F_s/2 > F_0$—then

$$X(e^{j\omega}) = \frac{1}{T_s} X\left(j\frac{\omega}{T_s}\right); \quad -\frac{\pi}{T_s} < \frac{\omega}{T_s} \le \frac{\pi}{T_s} \tag{3.29}$$

which leads to the sampling theorem for band-limited signals.

■ **THEOREM 3** ***Sampling Principle***

A band-limited signal $x_a(t)$ with bandwidth F_0 can be reconstructed from its sample values $x(n) = x_a(nT_s)$ if the sampling frequency $F_s = 1/T_s$ is greater than twice the bandwidth F_0 of $x_a(t)$.

$$F_s > 2F_0$$

Otherwise aliasing would result in $x(n)$. The sampling rate of $2F_0$ for an analog band-limited signal is called the Nyquist rate.

Note: After $x_a(t)$ is sampled, the highest analog frequency that $x(n)$ represents is $F_s/2$ Hz (or $\omega = \pi$). This agrees with the implication stated in property 2 of the discrete-time Fourier transform in Section 3.1. Before we delve into MATLAB implementation of sampling, we first consider sampling of sinusoidal signals and the resulting Fourier transform in the following example.

□ **EXAMPLE 3.17** The analog signal $x_a(t) = 4 + 2\cos(150\pi t + \pi/3) + 4\sin(350\pi t)$ is sampled at $F_s = 200$ sam/sec to obtain the discrete-time signal $x(n)$. Determine $x(n)$ and its corresponding DTFT $X(e^{j\omega})$.

Solution The highest frequency in the given $x_a(t)$ is $F_0 = 175$ Hz. Since $F_s = 200$, which is less than $2F_0$, there will be aliasing in $x(n)$ after sampling. The sampling interval is $T_s = 1/F_s = 0.005$ sec. Hence we have

$$\begin{aligned} x(n) &= x_a(nT_s) = x_a(0.005n) \\ &= 4 + 2\cos\left(0.75\pi n + \frac{\pi}{3}\right) + 4\sin(1.75\pi n) \end{aligned} \tag{3.30}$$

Note that the digital frequency, 1.75π, of the third term in (3.30) is outside the primary interval of $-\pi \le \omega \le \pi$, signifying that aliasing has occurred. From the periodicity property of digital sinusoidal sequences in Chapter 2, we know that the period of the digital sinusoid is 2π. Hence we can determine the alias of the frequency 1.75π. From (3.30) we have

$$\begin{aligned} x(n) &= 4 + 2\cos(0.75\pi n + \tfrac{\pi}{3}) + 4\sin(1.75\pi n - 2\pi n) \\ &= 4 + 2\cos(0.75\pi n + \tfrac{\pi}{3}) - 4\sin(0.25\pi n) \end{aligned} \tag{3.31}$$

Using Euler's identity, we can expess $x(n)$ as

$$x(n) = 4 + e^{j\pi/3}e^{j0.75\pi n} + e^{-j\pi/3}e^{-j0.75\pi n} + 2je^{j0.25\pi n} - 2je^{j0.25\pi n} \tag{3.32}$$

From Table 3.1 and the DTFT properties, the DTFT of $x(n)$ is given by

$$\begin{aligned} X(e^{j\omega}) \;=\; & 8\pi\delta(\omega) + 2\pi e^{j\pi/3}\delta(\omega - 0.75\pi) + 2\pi e^{-j\pi/3}\delta(\omega + 0.75\pi) \\ & + j4\pi\delta(\omega - 0.25\pi) - j4\pi\delta(\omega + 0.25\pi),\ -\pi \le \omega \le \pi. \end{aligned} \tag{3.33}$$

The plot of $X(e^{j\omega})$ is shown in Figure 3.15. □

3.4.2 MATLAB IMPLEMENTATION

In a strict sense it is not possible to analyze analog signals using MATLAB unless we use the Symbolic toolbox. However, if we sample $x_a(t)$ on a fine grid that has a sufficiently small time increment to yield a smooth plot and a large enough maximum time to show all the modes, then we can approximate its analysis. Let Δt be the grid interval such that $\Delta t \ll T_s$. Then

$$x_G(m) \triangleq x_a(m\Delta t) \tag{3.34}$$

can be used as an array to simulate an analog signal. The sampling interval T_s should not be confused with the grid interval Δt, which is used strictly to represent an analog signal in MATLAB. Similarly, the Fourier transform relation (3.24) should also be approximated in light of (3.34) as follows:

$$X_a(j\Omega) \approx \sum_m x_G(m)e^{-j\Omega m\Delta t}\Delta t = \Delta t \sum_m x_G(m)e^{-j\Omega m\Delta t} \tag{3.35}$$

Now if $x_a(t)$ [and hence $x_G(m)$] is of finite duration, then (3.35) is similar to the discrete-time Fourier transform relation (3.3) and hence can be implemented in MATLAB in a similar fashion to analyze the sampling phenomenon.

□ **EXAMPLE 3.18** Let $x_a(t) = e^{-1000|t|}$. Determine and plot its Fourier transform.

Solution From (3.24)

$$\begin{aligned} X_a(j\Omega) = \int_{-\infty}^{\infty} x_a(t)e^{-j\Omega t}dt \;&=\; \int_{-\infty}^{0} e^{1000t}e^{-j\Omega t}dt + \int_{0}^{\infty} e^{-1000t}e^{-j\Omega t}dt \\ &= \frac{0.002}{1 + (\frac{\Omega}{1000})^2} \end{aligned} \tag{3.36}$$

which is a real-valued function since $x_a(t)$ is a real and even signal. To evaluate $X_a(j\Omega)$ numerically, we have to first approximate $x_a(t)$ by a finite-duration grid sequence $x_G(m)$. Using the approximation $e^{-5} \approx 0$, we note that $x_a(t)$ can be approximated by a finite-duration signal over $-0.005 \le t \le 0.005$ (or equivalently, over $[-5, 5]$ msec). Similarly from (3.36), $X_a(j\Omega) \approx 0$ for $\Omega \ge 2\pi\,(2000)$. Hence choosing

$$\Delta t = 5 \times 10^{-5} \ll \frac{1}{2\,(2000)} = 25 \times 10^{-5}$$

we can obtain $x_G(m)$ and then implement (3.35) in MATLAB.

```
% Analog Signal
>> Dt = 0.00005; t = -0.005:Dt:0.005; xa = exp(-1000*abs(t));
% Continuous-time Fourier Transform
>>Wmax = 2*pi*2000; K = 500; k = 0:1:K; W = k*Wmax/K;
>>Xa = xa * exp(-j*t'*W) * Dt; Xa = real(Xa);
>>W = [-fliplr(W), W(2:501)]; % Omega from -Wmax to Wmax
>>Xa = [fliplr(Xa), Xa(2:501)]; % Xa over -Wmax to Wmax interval
>>subplot(2,1,1);plot(t*1000,xa);
>>xlabel('t in msec.'); ylabel('xa(t)')
>>title('Analog Signal')
>>subplot(2,1,2);plot(W/(2*pi*1000),Xa*1000);
>>xlabel('Frequency in KHz'); ylabel('Xa(jW)*1000')
>>title('Continuous-time Fourier Transform')
```

Figure 3.11 shows the plots of $x_a(t)$ and $X_a(j\Omega)$. Note that to reduce the number of computations, we computed $X_a(j\Omega)$ over $[0, 4000\pi]$ rad/sec (or equivalently, over $[0, 2]$ KHz) and then duplicated it over $[-4000\pi, 0]$ for plotting purposes. The displayed plot of $X_a(j\Omega)$ agrees with (3.36). □

□ **EXAMPLE 3.19** To study the effect of sampling on the frequency-domain quantities, we will sample $x_a(t)$ in Example 3.18 at 2 different sampling frequencies.

a. Sample $x_a(t)$ at $F_s = 5000$ sam/sec to obtain $x_1(n)$. Determine and plot $X_1(e^{j\omega})$.
b. Sample $x_a(t)$ at $F_s = 1000$ sam/sec to obtain $x_2(n)$. Determine and plot $X_2(e^{j\omega})$.

Solution

a. Since the bandwidth of $x_a(t)$ is 2KHz, the Nyquist rate is 4000 sam/sec, which is less than the given F_s. Therefore aliasing will be (almost) nonexistent.

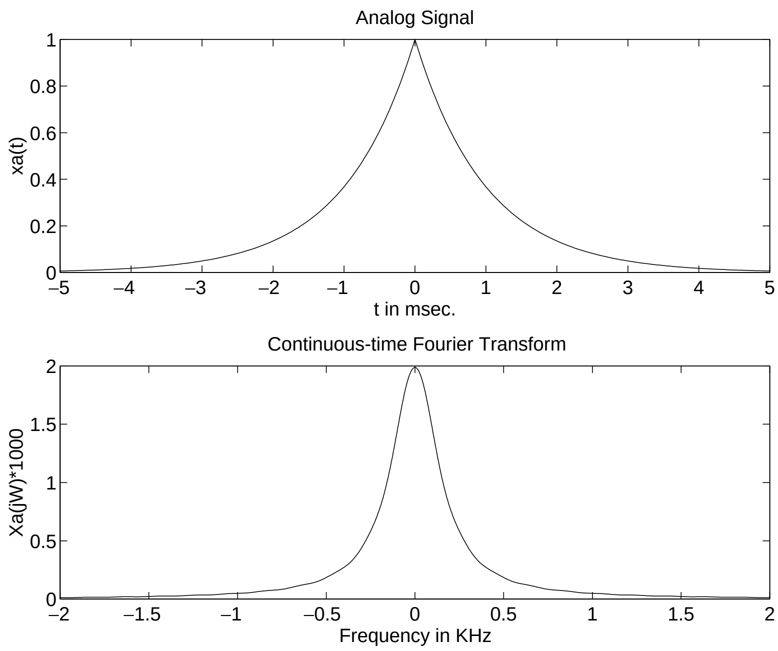


FIGURE 3.11 *Plots in Example 3.18*

MATLAB script:

```
% Analog Signal
>> Dt = 0.00005; t = -0.005:Dt:0.005; xa = exp(-1000*abs(t));
% Discrete-time Signal
>> Ts = 0.0002; n = -25:1:25; x = exp(-1000*abs(n*Ts));
% Discrete-time Fourier transform
>> K = 500; k = 0:1:K; w = pi*k/K;
>> X = x * exp(-j*n'*w); X = real(X);
>> w = [-fliplr(w), w(2:K+1)];   X = [fliplr(X), X(2:K+1)];
>> subplot(2,1,1);plot(t*1000,xa);
>> xlabel('t in msec.'); ylabel('x1(n)')
>> title('Discrete Signal'); hold on
>> stem(n*Ts*1000,x); gtext('Ts=0.2 msec'); hold off
>> subplot(2,1,2);plot(w/pi,X);
>> xlabel('Frequency in pi units'); ylabel('X1(w)')
>> title('Discrete-time Fourier Transform')
```

In the top plot in Figure 3.12, we have superimposed the discrete signal $x_1(n)$ over $x_a(t)$ to emphasize the sampling. The plot of $X_2(e^{j\omega})$ shows that it is a scaled version (scaled by $F_s = 5000$) of $X_a(j\Omega)$. Clearly there is no aliasing.

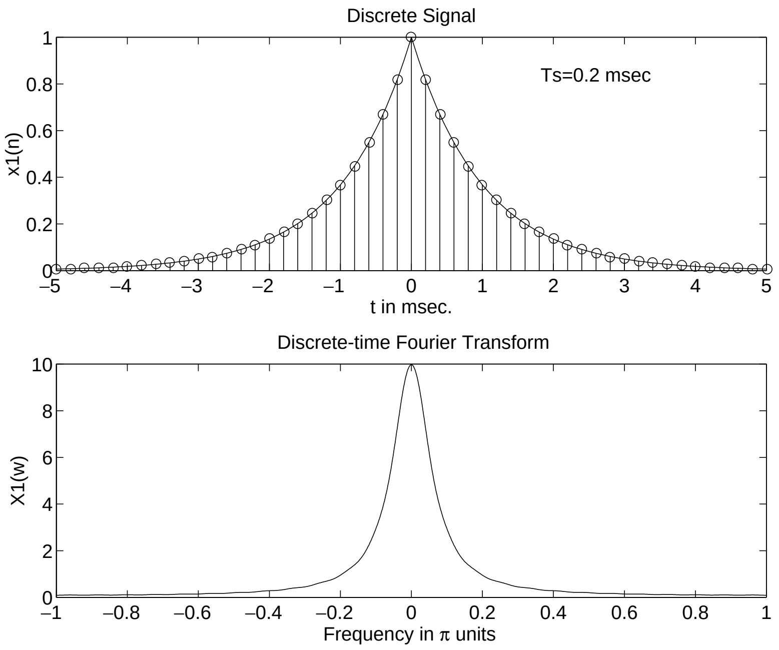


FIGURE 3.12 *Plots in Example 3.19a*

b. Here $F_s = 1000 < 4000$. Hence there will be a considerable amount of aliasing. This is evident from Figure 3.13, in which the shape of $X(e^{j\omega})$ is different from that of $X_a(j\Omega)$ and can be seen to be a result of adding overlapping replicas of $X_a(j\Omega)$. □

3.4.3 RECONSTRUCTION

From the sampling theorem and the preceding examples, it is clear that if we sample band-limited $x_a(t)$ above its Nyquist rate, then we can reconstruct $x_a(t)$ from its samples $x(n)$. This reconstruction can be thought of as a 2-step process:

- First the samples are converted into a weighted impulse train.

$$\sum_{n=-\infty}^{\infty} x(n)\delta(t-nT_s) = \cdots + x(-1)\delta(n+T_s) + x(0)\delta(t) + x(1)\delta(n-T_s) + \cdots$$

- Then the impulse train is filtered through an ideal analog lowpass filter band-limited to the $[-F_s/2, F_s/2]$ band.

$$\begin{tikzcd} x(n) \arrow[r] & \boxed{\text{Impulse train conversion}} \arrow[r] & \boxed{\text{Ideal lowpass filter}} \arrow[r] & x_a(t) \end{tikzcd}$$

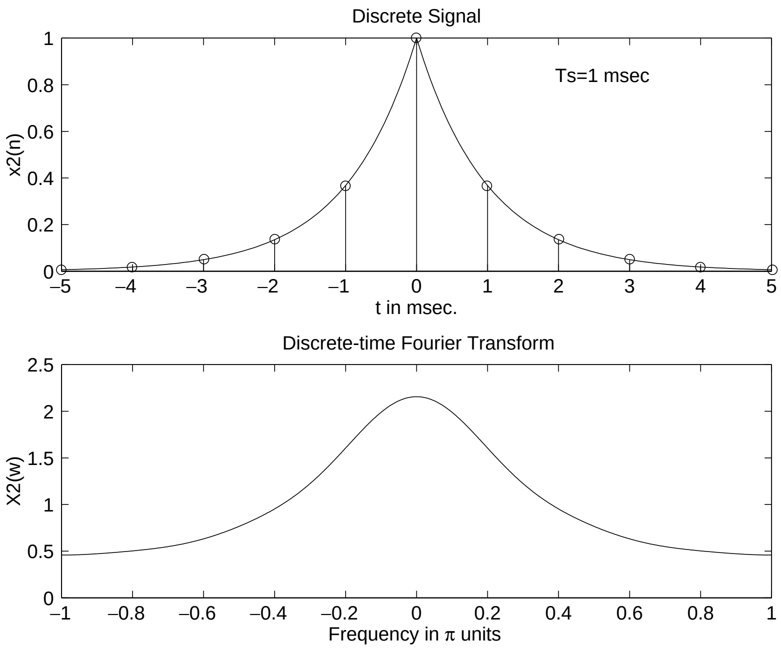


FIGURE 3.13 *Plots in Example 3.18b*

This two-step procedure can be described mathematically using an interpolating formula [23]

$$x_a(t) = \sum_{n=-\infty}^{\infty} x(n)\,\mathrm{sinc}\left[F_s(t - nT_s)\right] \tag{3.37}$$

where $\mathrm{sinc}(x) = \frac{\sin \pi x}{\pi x}$ is an interpolating function. The physical interpretation of the above reconstruction (3.37) is given in Figure 3.14, from which we observe that this *ideal* interpolation is not practically feasible because the entire system is noncausal and hence not realizable.

☐ **EXAMPLE 3.20** Consider the sampled signal $x(n)$ from Example 3.17. It is applied as an input to an ideal D/A converter (that is, an ideal interpolator) to obtain the analog signal $y_a(t)$. The ideal D/A converter is also operating at $F_s = 200$ sam/sec. Obtain the reconstructed signal $y_a(t)$, and determine whether the sampling/reconstruction operation resulted in any aliasing. Also plot the Fourier transforms $X_a(j\Omega)$, $X(e^{j\omega})$, and $Y_a(j\Omega)$.

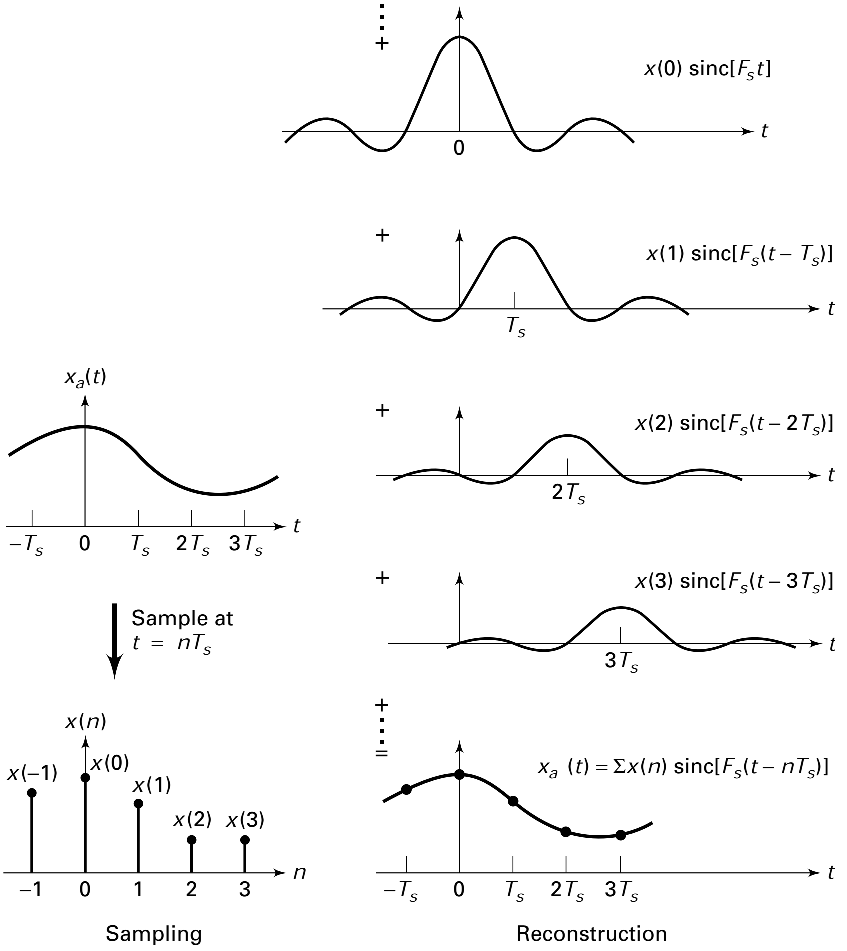


FIGURE 3.14 *Reconstruction of band-limited signal from its samples*

Solution We can determine $y_a(t)$ using (3.31). However, since all frequencies in the sinusoidal sequence $x(n)$ are between the primary period of $-\pi \le \omega \le \pi$, we can equivalently obtain $y_a(t)$ by substituting n by tF_s. Thus from (3.31), we have

$$\begin{aligned} y_a(t) &= x(n)\Big|_{n=tFs} = x(n)\Big|_{n=200t} \\ &= 4 + 2\cos\left(0.75\pi 200t + \frac{\pi}{3}\right) - 4\sin(0.25\pi 200t) \\ &= 4 + 2\cos\left(150\pi t + \frac{\pi}{3}\right) - 4\sin(50\pi t) \qquad \textbf{(3.38)} \end{aligned}$$

As expected, the 175 Hz component in $x_a(t)$ is aliased into the 25 Hz component in $y_a(t)$.

Using Euler's identity on the given $x_a(t)$ and the properties, the CTFT $X_a(j\Omega)$ is given by

$$\begin{aligned} X_a(j\Omega) &= 8\pi\delta(\Omega) + 2\pi e^{j\pi/3}\delta(\Omega - 150\pi) + 2\pi e^{-j\pi/3}\delta(\Omega + 150\pi) \\ &\quad + 4j\pi\delta(\Omega - 350\pi) - 4j\pi\delta(\Omega + 350\pi). \end{aligned} \tag{3.39}$$

It is informative to plot the CTFT $X_a(j\Omega)$ as a function of the cyclic frequency F in Hz using $\Omega = 2\pi F$. Thus the quantity $X_a(j2\pi F)$ from (3.39) is given by

$$\begin{aligned} X_a(j2\pi F) &= 4\delta(F) + e^{j\pi/3}\delta(F - 75) + e^{-j\pi/3}\delta(F + 75) \\ &\quad + 2j\delta(F - 175) - 2j\delta(F + 175). \end{aligned} \tag{3.40}$$

where we have used the identity $\delta(\Omega) = \delta(2\pi F) = \frac{1}{2\pi}\delta(F)$. Similarly, the CTFT $Y_a(j2\pi F)$ is given by

$$\begin{aligned} Y_a(j2\pi F) &= 4\delta(F) + e^{j\pi/3}\delta(F - 75) + e^{-j\pi/3}\delta(F + 75) \\ &\quad + 2j\delta(F - 25) - 2j\delta(F + 25). \end{aligned} \tag{3.41}$$

Figure 3.15a shows the CTFT of the original signal $x_a(t)$ as a function of F. The DTFT $X\left(e^{j\omega}\right)$ of the sampled sequence $x(n)$ is shown as a function of ω in Figure 3.15b, in which the impulses due to shifted replicas are shown in gray shade for clarity. The ideal D/A converter response is also shown in gray shade. The CTFT of the reconstructed signal $y_a(t)$ is shown in Figure 3.15c which clearly shows the aliasing effect. □

Practical D/A converters In practice we need a different approach than (3.37). The two-step procedure is still feasible, but now we replace the ideal lowpass filter by a practical analog lowpass filter. Another interpretation of (3.37) is that it is an infinite-order interpolation. We want finite-order (and in fact low-order) interpolations. There are several approaches to do this.

- **Zero-order-hold (ZOH) interpolation:** In this interpolation a given sample value is held for the sample interval until the next sample is received.

  $$\hat{x}_a(t) = x(n), \quad nT_s \le n < (n+1)T_s$$

  which can be obtained by filtering the impulse train through an interpolating filter of the form

  $$h_0(t) = \begin{cases} 1, & 0 \le t \le T_s \\ 0, & \text{otherwise} \end{cases}$$

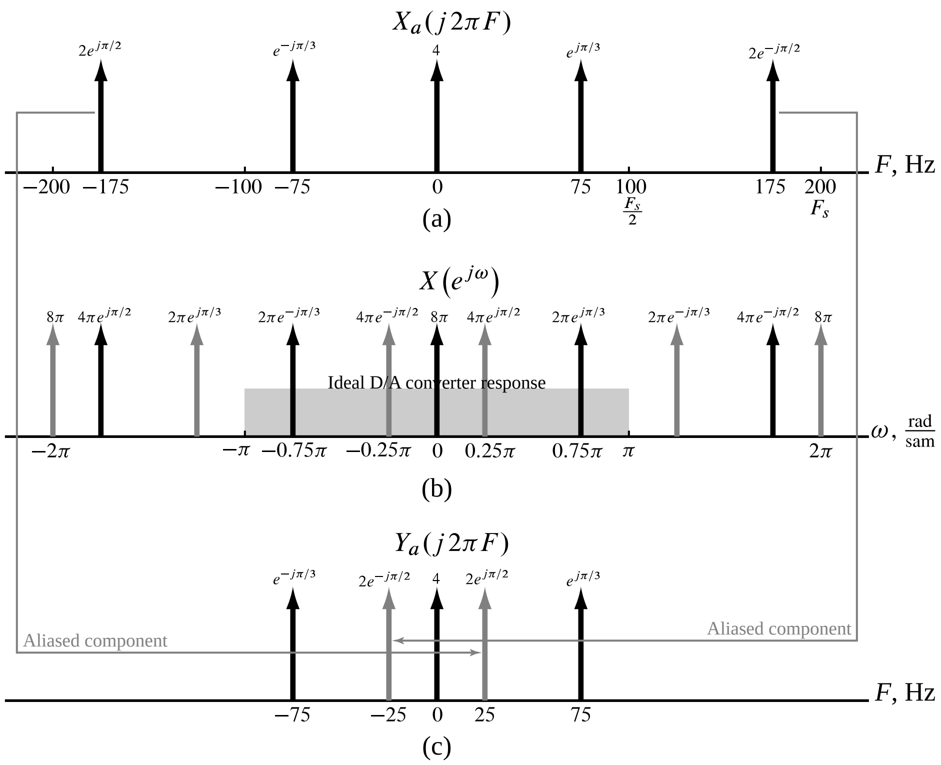


FIGURE 3.15 *Fourier transforms of the sinusoidal signals $x_a(t)$, $x(n)$, and $y_a(t)$*

which is a rectangular pulse. The resulting signal is a piecewise-constant (staircase) waveform which requires an appropriately designed analog postfilter for accurate waveform reconstruction.

$$x(n) \longrightarrow \boxed{\text{ZOH}} \longrightarrow \hat{x}_a(t) \longrightarrow \boxed{\text{Postfilter}} \longrightarrow x_a(t)$$

- **1st-order-hold (FOH) interpolation:** In this case the adjacent samples are joined by straight lines. This can be obtained by filtering the impulse train through

$$h_1(t) = \begin{cases} 1 + \dfrac{t}{T_s}, & 0 \le t \le T_s \\ 1 - \dfrac{t}{T_s}, & T_s \le t \le 2T_s \\ 0, & \text{otherwise} \end{cases}$$

Once again, an appropriately designed analog postfilter is required for accurate reconstruction. These interpolations can be extended to higher orders. One particularly useful interpolation employed by MATLAB is the following.

- **Cubic spline interpolation:** This approach uses spline interpolants for a smoother, but not necessarily more accurate, estimate of the analog signals between samples. Hence this interpolation does not require an analog postfilter. The smoother reconstruction is obtained by using a set of piecewise continuous third-order polynomials called *cubic splines*, given by [3]

$$\begin{aligned} x_a(t) = {} & \alpha_0(n) + \alpha_1(n)(t - nT_s) + \alpha_2(n)(t - nT_s)^2 \\ & + \alpha_3(n)(t - nT_s)^3, \quad nT_s \le n < (n+1)T_s \end{aligned} \tag{3.42}$$

where $\{\alpha_i(n), 0 \le i \le 3\}$ are the polynomial coefficients, which are determined by using least-squares analysis on the sample values. (Strictly speaking, this is not a causal operation but is a convenient one in MATLAB.)

3.4.4 MATLAB IMPLEMENTATION

For interpolation between samples MATLAB provides several approaches. The function `sinc(x)`, which generates the $(\sin \pi x)/\pi x$ function, can be used to implement (3.37), given a finite number of samples. If $\{x(n),\, n_1 \le n \le n_2\}$ is given, and if we want to interpolate $x_a(t)$ on a very fine grid with the grid interval Δt, then from (3.37)

$$x_a(m\Delta t) \approx \sum_{n=n_1}^{n_2} x(n)\,\text{sinc}\left[F_s(m\Delta t - nT_s)\right], \quad t_1 \le m\Delta t \le t_2 \tag{3.43}$$

which can be implemented as a matrix-vector multiplication operation as shown below.

```
>> n = n1:n2; t = t1:t2; Fs = 1/Ts; nTs = n*Ts;  % Ts is the sampling interval
 >> xa = x * sinc(Fs*(ones(length(n),1)*t-nTs'*ones(1,length(t))));
```

Note that it is not possible to obtain an *exact* analog $x_a(t)$ in light of the fact that we have assumed a finite number of samples. We now demonstrate the use of the `sinc` function in the following two examples and also study the aliasing problem in the time domain.

□ **EXAMPLE 3.21** From the samples $x_1(n)$ in Example 3.19a, reconstruct $x_a(t)$ and comment on the results.

Solution Note that $x_1(n)$ was obtained by sampling $x_a(t)$ at $T_s = 1/F_s = 0.0002$ sec. We will use the grid spacing of 0.00005 sec over $-0.005 \le t \le 0.005$, which gives $x(n)$ over $-25 \le n \le 25$.

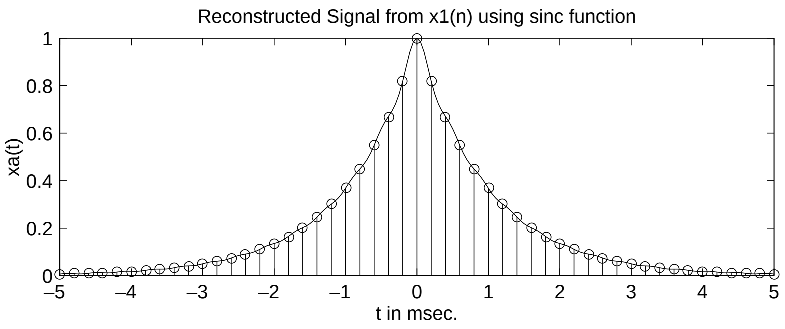


FIGURE 3.16 *Reconstructed signal in Example 3.21*

MATLAB script:

```
% Discrete-time Signal x1(n)
>> Ts = 0.0002; n = -25:1:25; nTs = n*Ts;  x = exp(-1000*abs(nTs));
% Analog Signal reconstruction
>> Dt = 0.00005; t = -0.005:Dt:0.005;
>> xa = x * sinc(Fs*(ones(length(n),1)*t-nTs'*ones(1,length(t))));
% check
>> error = max(abs(xa - exp(-1000*abs(t))))
error =
     0.0363
```

The maximum error between the reconstructed and the actual analog signal is 0.0363, which is due to the fact that $x_a(t)$ is not strictly band-limited (and also we have a finite number of samples). From Figure 3.16, we note that visually the reconstruction is excellent. □

□ **EXAMPLE 3.22** From the samples $x_2(n)$ in Example 3.17b reconstruct $x_a(t)$ and comment on the results.

Solution In this case $x_2(n)$ was obtained by sampling $x_a(t)$ at $T_s = 1/F_s = 0.001$ sec. We will again use the grid spacing of 0.00005 sec over $-0.005 \leq t \leq 0.005$, which gives $x(n)$ over $-5 \leq n \leq 5$.

```
% Discrete-time Signal x2(n)
>> Ts = 0.001; n = -5:1:5; nTs = n*Ts;  x = exp(-1000*abs(nTs));
% Analog Signal reconstruction
>> Dt = 0.00005; t = -0.005:Dt:0.005;
>> xa = x * sinc(Fs*(ones(length(n),1)*t-nTs'*ones(1,length(t))));
% check
>> error = max(abs(xa - exp(-1000*abs(t))))
error =
     0.1852
```

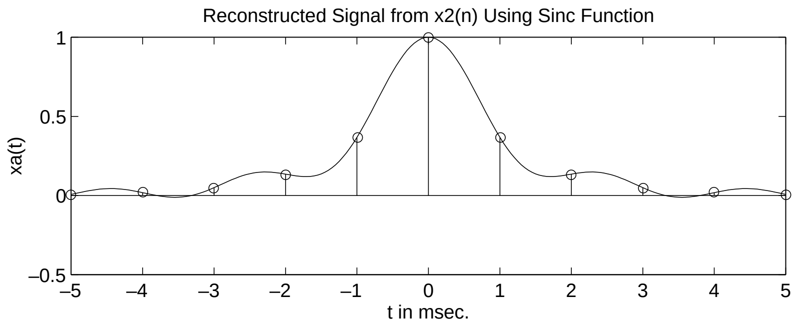


FIGURE 3.17 *Reconstructed signal in Example 3.22*

The maximum error between the reconstructed and the actual analog signals is 0.1852, which is significant and cannot be attributed to the nonband-limitedness of $x_a(t)$ alone. From Figure 3.17, observe that the reconstructed signal differs from the actual one in many places over the interpolated regions. This is the visual demonstration of aliasing in the time domain. □

The second MATLAB approach for signal reconstruction is a plotting approach. The `stairs` function plots a staircase (ZOH) rendition of the analog signal, given its samples, while the `plot` function depicts a linear (FOH) interpolation between samples.

□ **EXAMPLE 3.23** Plot the reconstructed signal from the samples $x_1(n)$ in Example 3.19 using the ZOH and the FOH interpolations. Comment on the plots.

Solution Note that in this reconstruction we do not compute $x_a(t)$ but merely plot it using its samples.

```
% Discrete-time Signal x1(n) : Ts = 0.0002
>> Ts = 0.0002; n = -25:1:25; nTs = n*Ts;  x = exp(-1000*abs(nTs));
% Plots
>> subplot(2,1,1); stairs(nTs*1000,x);
>> xlabel('t in msec.'); ylabel('xa(t)')
>> title('Reconstructed Signal from x1(n) using zero-order-hold'); hold on
>> stem(n*Ts*1000,x); hold off
%
% Discrete-time Signal x2(n) : Ts = 0.001
>> Ts = 0.001; n = -5:1:5; nTs = n*Ts;  x = exp(-1000*abs(nTs));
% Plots
>> subplot(2,1,2); plot(nTs*1000,x);
>> xlabel('t in msec.'); ylabel('xa(t)')
>> title('Reconstructed Signal from x2(n) using zero-order-hold'); hold on
>> stem(n*Ts*1000,x); hold off
```

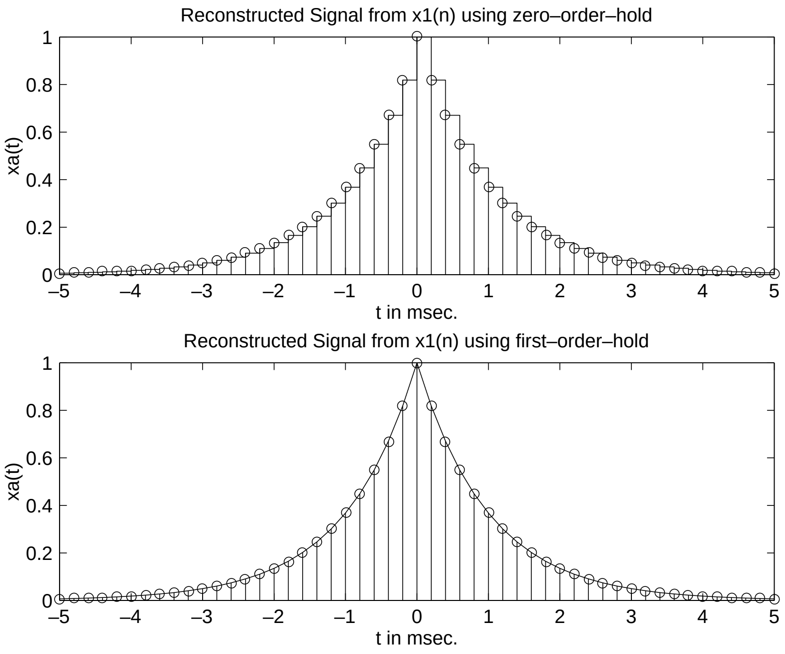


FIGURE 3.18 *Signal reconstruction in Example 3.23*

The plots are shown in Figure 3.18, from which we observe that the ZOH reconstruction is a crude one and that the further processing of analog signal is necessary. The FOH reconstruction appears to be a good one, but a careful observation near $t = 0$ reveals that the peak of the signal is not correctly reproduced. In general, if the sampling frequency is much higher than the Nyquist rate, then the FOH interpolation provides an acceptable reconstruction. □

The third approach of reconstruction in MATLAB involves the use of cubic spline functions. The `spline` function implements interpolation between sample points. It is invoked by `xa = spline(nTs,x,t)`, in which `x` and `nTs` are arrays containing samples $x(n)$ at nT_s instances, respectively, and `t` array contains a fine grid at which $x_a(t)$ values are desired. Note once again that it is not possible to obtain an *exact* analog $x_a(t)$.

□ **EXAMPLE 3.24** From the samples $x_1(n)$ and $x_2(n)$ in Example 3.19, reconstruct $x_a(t)$ using the `spline` function. Comment on the results.

Solution This example is similar to Examples 3.21 and 3.22. Hence sampling parameters are the same as before.

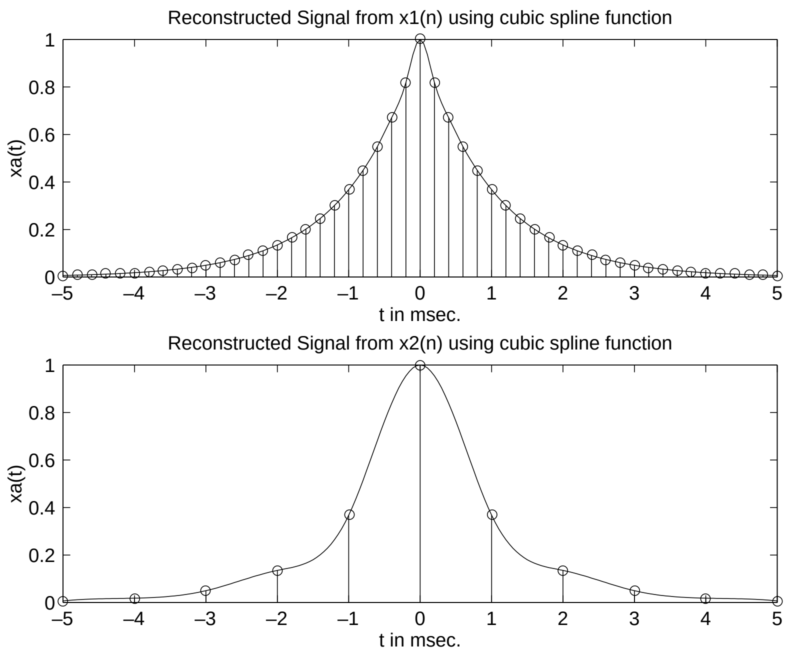


FIGURE 3.19 *Reconstructed signal in Example 3.24*

MATLAB script:

```
% a) Discrete-time Signal x1(n): Ts = 0.0002
>> Ts = 0.0002; n = -25:1:25; nTs = n*Ts;   x = exp(-1000*abs(nTs));
% Analog Signal reconstruction
>> Dt = 0.00005; t = -0.005:Dt:0.005;   xa = spline(nTs,x,t);
% check
>> error = max(abs(xa - exp(-1000*abs(t))))
error = 0.0317
```

The maximum error between the reconstructed and the actual analog signal is 0.0317, which is due to the nonideal interpolation and the fact that $x_a(t)$ is nonband-limited. Comparing this error with that from the sinc (or ideal) interpolation, we note that this error is lower. The ideal interpolation generally suffers more from time-limitedness (or from a finite number of samples). From the top plot in Figure 3.19 we observe that visually the reconstruction is excellent.

MATLAB script:

```
% Discrete-time Signal x2(n): Ts = 0.001
>> Ts = 0.001; n = -5:1:5; nTs = n*Ts;   x = exp(-1000*abs(nTs));
% Analog Signal reconstruction
>> Dt = 0.00005; t = -0.005:Dt:0.005;   xa = spline(nTs,x,t);
```

```
% check
>> error = max(abs(xa - exp(-1000*abs(t))))
error = 0.1679
```

The maximum error in this case is 0.1679, which is significant and cannot be attributed to the nonideal interpolation or nonband-limitedness of $x_a(t)$. From the bottom plot in Figure 3.19 observe that the reconstructed signal again differs from the actual one in many places over the interpolated regions. □

From these examples it is clear that for practical purposes the `spline` interpolation provides the best results.

3.5 PROBLEMS

P3.1 Using the matrix-vector multiplication approach discussed in this chapter, write a MATLAB function to compute the DTFT of a finite-duration sequence. The format of the function should be

```
function [X] = dtft(x,n,w)
% Computes Discrete-time Fourier Transform
% [X] = dtft(x,n,w)
%   X = DTFT values computed at w frequencies
%   x = finite duration sequence over n
%   n = sample position vector
%   w = frequency location vector
```

Use this function to compute the DTFT $X(e^{j\omega})$ of the following finite-duration sequences over $-\pi \le \omega \le \pi$. Plot DTFT magnitude and angle graphs in one figure window.

1. $x(n) = (0.6)^{|n|}\,[u(n+10) - u(n-11)]$. Comment on the angle plot.
2. $x(n) = n(0.9)^n\,[u(n) - u(n-21)]$.
3. $x(n) = [\cos(0.5\pi n) + j\sin(0.5\pi n)][u(n) - u(n-51)]$. Comment on the magnitude plot.
4. $x(n) = \{\underset{\uparrow}{4}, 3, 2, 1, 1, 2, 3, 4\}$. Comment on the angle plot.
5. $x(n) = \{\underset{\uparrow}{4}, 3, 2, 1, -1, -2, -3, -4\}$. Comment on the angle plot.

P3.2 Let $x_1(n) = \{\underset{\uparrow}{1}, 2, 2, 1\}$. A new sequence $x_2(n)$ is formed using

$$x_2(n) = \begin{cases} x_1(n), & 0 \le n \le 3; \\ x_1(n-4), & 4 \le n \le 7; \\ 0, & \text{Otherwise.} \end{cases} \tag{3.44}$$

1. Express $X_2(e^{j\omega})$ in terms of $X_1(e^{j\omega})$ without explicitly computing $X_1(e^{j\omega})$.
2. Verify your result using MATLAB by computing and plotting magnitudes of the respective DTFTs.

P3.3 Determine analytically the DTFT of each of the following sequences. Plot the magnitude and angle of $X(e^{j\omega})$ over $0 \le \omega \le \pi$.

1. $x(n) = 2\,(0.5)^n\, u(n+2)$.
2. $x(n) = (0.6)^{|n|}\,[u(n+10) - u(n-11)]$.
3. $x(n) = n\,(0.9)^n\, u(n+3)$.
4. $x(n) = (n+3)\,(0.8)^{n-1}\, u(n-2)$.
5. $x(n) = 4\,(-0.7)^n \cos(0.25\pi n) u(n)$.

P3.4 The following finite-duration sequences are called *windows* and are very useful in DSP.

$$\begin{aligned}
\text{Rectangular: } \mathcal{R}_M(n) &= \begin{cases} 1, & 0 \le n < M \\ 0, & \text{otherwise} \end{cases}; \\
\text{Hanning: } \mathcal{C}_M(n) &= 0.5\left[1 - \cos\frac{2\pi n}{M-1}\right]\mathcal{R}_M(n) \\
\text{Triangular: } \mathcal{T}_M(n) &= \left[1 - \frac{|M-1-2n|}{M-1}\right]\mathcal{R}_M(n); \\
\text{Hamming: } \mathcal{H}_M(n) &= \left[0.54 - 0.46\cos\frac{2\pi n}{M-1}\right]\mathcal{R}_M(n)
\end{aligned}$$

For each of these windows, determine their DTFTs for $M = 10,\ 25,\ 50,\ 101$. Scale transform values so that the maximum value is equal to 1. Plot the magnitude of the normalized DTFT over $-\pi \le \omega \le \pi$. Study these plots and comment on their behavior as a function of M.

P3.5 Using the definition of the DTFT in (3.1), determine the sequences corresponding to the following DTFTs:

1. $X(e^{j\omega}) = 3 + 2\cos(\omega) + 4\cos(2\omega)$.
2. $X(e^{j\omega}) = [1 - 6\cos(3\omega) + 8\cos(5\omega)]\, e^{-j3\omega}$.
3. $X(e^{j\omega}) = 2 + j4\sin(2\omega) - 5\cos(4\omega)$.
4. $X(e^{j\omega}) = [1 + 2\cos(\omega) + 3\cos(2\omega)]\cos(\omega/2)e^{-j5\omega/2}$.
5. $X(e^{j\omega}) = j\,[3 + 2\cos(\omega) + 4\cos(2\omega)]\sin(\omega)e^{-j3\omega}$.

P3.6 Using the definition of the inverse DTFT in (3.2), determine the sequences corresponding to the following DTFTs:

1. $X(e^{j\omega}) = \begin{cases} 1, & 0 \le |\omega| \le \pi/3; \\ 0, & \pi/3 < |\omega| \le \pi. \end{cases}$
2. $X(e^{j\omega}) = \begin{cases} 0, & 0 \le |\omega| \le 3\pi/4; \\ 1, & 3\pi/4 < |\omega| \le \pi. \end{cases}$
3. $X(e^{j\omega}) = \begin{cases} 2, & 0 \le |\omega| \le \pi/8; \\ 1, & \pi/8 < |\omega| \le 3\pi/4. \\ 0, & 3\pi/4 < |\omega| \le \pi. \end{cases}$
4. $X(e^{j\omega}) = \begin{cases} 0, & -\pi \le |\omega| < \pi/4; \\ 1, & \pi/4 \le |\omega| \le 3\pi/4. \\ 0, & 3\pi/4 < |\omega| \le \pi. \end{cases}$
5. $X(e^{j\omega}) = \omega\, e^{j(\pi/2 - 10\omega)}$.

Remember that the above transforms are periodic in ω with period equal to 2π. Hence, functions are given only over the primary period of $-\pi \leq \omega \leq \pi$.

P3.7 A complex-valued sequence $x(n)$ can be decomposed into a conjugate symmetric part $x_e(n)$ and an conjugate anti-symmetric part $x_o(n)$ as discussed in Chapter 2. Show that

$$\mathcal{F}\left[x_e(n)\right] = X_R(e^{j\omega}) \quad \text{and} \quad \mathcal{F}\left[x_o(n)\right] = jX_I(e^{j\omega})$$

where $X_R(e^{j\omega})$ and $X_R(e^{j\omega})$ are the real and imaginary parts of the DTFT $X(e^{j\omega})$ respectively. Verify this property on

$$x(n) = 2(0.9)^{-n}\left[\cos(0.1\pi n) + j\sin(0.9\pi n)\right]\left[u(n) - u(n-10)\right]$$

using the MATLAB functions developed in Chapter 2.

P3.8 A complex-valued DTFT $X(e^{j\omega})$ can also be decomposed into its conjugate symmetric part $X_e(e^{j\omega})$ and conjugate anti-symmetric part $X_o(e^{j\omega})$, i.e.,

$$X(e^{j\omega}) = X_e(e^{j\omega}) + X_o(e^{j\omega})$$

where

$$X_e(e^{j\omega}) = \frac{1}{2}[X(e^{j\omega}) + X^*(e^{-j\omega})] \quad \text{and} \quad X_0(e^{j\omega}) = \frac{1}{2}[X(e^{j\omega}) - X^*(e^{-j\omega})]$$

Show that

$$\mathcal{F}^{-1}[X_e(e^{j\omega})] = x_R(n) \quad \text{and} \quad \mathcal{F}^{-1}[X_0(e^{j\omega})] = jx_I(n)$$

where $x_R(n)$ and $x_I(n)$ are the real and imaginary parts of $x(n)$. Verify this property on

$$x(n) = e^{j0.1\pi n}\left[u(n) - u\left(n-20\right)\right]$$

using the MATLAB functions developed in Chapter 2.

P3.9 Using the frequency-shifting property of the DTFT, show that the real part of $X(e^{j\omega})$ of a sinusoidal pulse

$$x(n) = (\cos\omega_o n)\mathcal{R}_M(n)$$

where $\mathcal{R}_M(n)$ is the rectangular pulse given in Problem P3.4 is given by

$$\begin{aligned} X_{\text{R}}(e^{j\omega}) \;=\; & \frac{1}{2}\cos\left\{\frac{(\omega-\omega_0)(M-1)}{2}\right\}\frac{\sin\left\{(\omega-\omega_0)\,M/2\right\}}{\sin\left\{(\omega-\omega_0)\,/2\right\}} \\ & +\frac{1}{2}\cos\left\{\frac{(\omega+\omega_0)(M-1)}{2}\right\}\frac{\sin\left\{[\omega-(2\pi-\omega_0)]\,M/2\right\}}{\sin\left\{[\omega-(2\pi-\omega_0)]\,/2\right\}} \end{aligned}$$

Compute and plot $X_{\text{R}}(e^{j\omega})$ for $\omega_o = \pi/2$ and $M = 5$, 15, 25, 100. Use the plotting interval $[-\pi, \pi]$. Comment on your results.

P3.10 Let $x(n) = \mathcal{T}_{10}(n)$ be a triangular pulse given in Problem P3.4. Using properties of the DTFT, determine and plot the DTFT of the following sequences.

1. $x(n) = \mathcal{T}_{10}(-n)$
2. $x(n) = \mathcal{T}_{10}(n) - \mathcal{T}_{10}(n-10)$
3. $x(n) = \mathcal{T}_{10}(n) * \mathcal{T}_{10}(-n)$
4. $x(n) = \mathcal{T}_{10}(n)e^{j\pi n}$
5. $x(n) = \cos(0.1\pi n)\mathcal{T}_{10}(n)$

P3.11 For each of the linear, shift-invariant systems described by the impulse response, determine the frequency response function $H(e^{j\omega})$. Plot the magnitude response $|H(e^{j\omega})|$ and the phase response $\angle H(e^{j\omega})$ over the interval $[-\pi, \pi]$.

1. $h(n) = (0.9)^{|n|}$
2. $h(n) = \text{sinc}(0.2n)[u(n+20) - u(n-20)]$, where $\text{sinc}\, 0 = 1$.
3. $h(n) = \text{sinc}(0.2n)[u(n) - u(n-40)]$
4. $h(n) = [(0.5)^n + (0.4)^n]u(n)$
5. $h(n) = (0.5)^{|n|}\cos(0.1\pi n)$

P3.12 Let $x(n) = A\cos(\omega_0 n + \theta_0)$ be an input sequence to an LTI system described by the impulse response $h(n)$. Show that the output sequence $y(n)$ is given by

$$y(n) = A|H(e^{j\omega_0})|\cos[\omega_0 n + \theta_0 + \angle H(e^{j\omega_0})]$$

P3.13 Let $x(n) = 3\cos(0.5\pi n + 60^\circ) + 2\sin(0.3\pi n)$ be the input to each of the systems described in Problem P3.11. In each case, determine the output sequence $y(n)$.

P3.14 An ideal lowpass filter is described in the frequency domain by

$$H_d(e^{j\omega}) = \begin{cases} 1 \cdot e^{-j\alpha\omega}, & |\omega| \le \omega_c \\ 0, & \omega_c < |\omega| \le \pi \end{cases}$$

where ω_c is called the cutoff frequency and α is called the phase delay.

1. Determine the ideal impulse response $h_d(n)$ using the IDTFT relation (3.2).
2. Determine and plot the truncated impulse response

$$h(n) = \begin{cases} h_d(n), & 0 \le n \le N-1 \\ 0, & \text{otherwise} \end{cases}$$

   for $N = 41$, $\alpha = 20$, and $\omega_c = 0.5\pi$.
3. Determine and plot the frequency response function $H(e^{j\omega})$, and compare it with the ideal lowpass filter response $H_d(e^{j\omega})$. Comment on your observations.

P3.15 An ideal highpass filter is described in the frequency-domain by

$$H_d(e^{j\omega}) = \begin{cases} 1 \cdot e^{-j\alpha\omega}, & \omega_c < |\omega| \le \pi \\ 0, & |\omega| \le \omega_c \end{cases}$$

where ω_c is called the cutoff frequency and α is called the phase delay.

1. Determine the ideal impulse response $h_d(n)$ using the IDTFT relation (3.2).
2. Determine and plot the truncated impulse response

$$h(n) = \begin{cases} h_d(n), & 0 \le n \le N-1 \\ 0, & \text{otherwise} \end{cases}$$

   for $N = 31$, $\alpha = 15$, and $\omega_c = 0.5\pi$.
3. Determine and plot the frequency response function $H(e^{j\omega})$, and compare it with the ideal highpass filter response $H_d(e^{j\omega})$. Comment on your observations

P3.16 For a linear, shift-invariant system described by the difference equation

$$y(n) = \sum_{m=0}^{M} b_m x(n-m) - \sum_{\ell=1}^{N} a_\ell y(n-\ell)$$

the frequency-response function is given by

$$H(e^{j\omega}) = \frac{\sum_{m=0}^{M} b_m e^{-j\omega m}}{1 + \sum_{\ell=1}^{N} a_\ell e^{-j\omega \ell}}$$

Write a MATLAB function `freqresp` to implement this relation. The format of this function should be

```
function [H] = freqresp(b,a,w)
% Frequency response function from difference equation
%  [H] = freqresp(b,a,w)
% H = frequency response array evaluated at w frequencies
% b = numerator coefficient array
% a = denominator coefficient array (a(1)=1)
% w = frequency location array
```

P3.17 Determine $H(e^{j\omega})$, and plot its magnitude and phase for each of the following systems:

1. $y(n) = \frac{1}{5}\sum_{m=0}^{4} x(n-m)$
2. $y(n) = x(n) - x(n-2) + 0.95y(n-1) - 0.9025y(n-2)$
3. $y(n) = x(n) - x(n-1) + x(n-2) + 0.95y(n-1) - 0.9025y(n-2)$
4. $y(n) = x(n) - 1.7678x(n-1) + 1.5625x(n-2) + 1.1314y(n-1) - 0.64y(n-2)$
5. $y(n) = x(n) - \sum_{\ell=1}^{5} (0.5)^\ell y(n-\ell)$

P3.18 A linear, shift-invariant system is described by the difference equation

$$y(n) = \sum_{m=0}^{3} x(n-2m) - \sum_{\ell=1}^{3} (0.81)^\ell y(n-2\ell)$$

Determine the steady-state response of the system to the following inputs:

1. $x(n) = 5 + 10(-1)^n$
2. $x(n) = 1 + \cos(0.5\pi n + \pi/2)$
3. $x(n) = 2\sin(\pi n/4) + 3\cos(3\pi n/4)$
4. $x(n) = \sum_{k=0}^{5} (k+1)\cos(\pi k n/4)$
5. $x(n) = \cos(\pi n)$

In each case, generate $x(n)$, $0 \le n \le 200$, and process it through the `filter` function to obtain $y(n)$. Compare your $y(n)$ with the steady-state responses in each case.

P3.19 An analog signal $x_a(t) = \sin(1000\pi t)$ is sampled using the following sampling intervals. In each case, plot the spectrum of the resulting discrete-time signal.

1. $T_s = 0.1$ ms
2. $T_s = 1$ ms
3. $T_s = 0.01$ sec

P3.20 We implement the following analog filter using a discrete filter.

$$x_a(t) \longrightarrow \boxed{\text{A/D}} \xrightarrow{x(n)} \boxed{h(n)} \xrightarrow{y(n)} \boxed{\text{D/A}} \longrightarrow y_a(t)$$

The sampling rate in the A/D and D/A is 8000 sam/sec, and the impulse response is $h(n) = (-0.9)^n u(n)$.

1. What is the digital frequency in $x(n)$ if $x_a(t) = 10\cos(10,000\pi t)$?
2. Determine the steady-state output $y_a(t)$ if $x_a(t) = 10\cos(10,000\pi t)$.
3. Determine the steady-state output $y_a(t)$ if $x_a(t) = 5\sin(8,000\pi t)$.
4. Find two other analog signals $x_a(t)$, with different analog frequencies, that will give the same steady-state output $y_a(t)$ when $x_a(t) = 10\cos(10,000\pi t)$ is applied.
5. To prevent aliasing, a prefilter would be required to process $x_a(t)$ before it passes to the A/D converter. What type of filter should be used, and what should be the largest cutoff frequency that would work for the given configuration?

P3.21 Consider an analog signal $x_a(t) = \cos(20\pi t)$, $0 \le t \le 1$. It is sampled at $T_s = 0.01$, 0.05, and 0.1 sec intervals to obtain $x(n)$.

1. For each T_s plot $x(n)$.
2. Reconstruct the analog signal $y_a(t)$ from the samples $x(n)$ using the sinc interpolation (use $\Delta t = 0.001$) and determine the frequency in $y_a(t)$ from your plot. (Ignore the end effects.)
3. Reconstruct the analog signal $y_a(t)$ from the samples $x(n)$ using the cubic spline interpolation, and determine the frequency in $y_a(t)$ from your plot. (Again, ignore the end effects.)
4. Comment on your results.

P3.22 Consider the analog signal $x_a(t) = \cos(20\pi t + \theta)$, $0 \le t \le 1$. It is sampled at $T_s = 0.05$ sec intervals to obtain $x(n)$. Let $\theta = 0$, $\pi/6$, $\pi/4$, $\pi/3$, $\pi/2$. For each of these θ values, perform the following.

1. Plot $x_a(t)$ and superimpose $x(n)$ on it using the `plot(n,x,'o')` function.
2. Reconstruct the analog signal $y_a(t)$ from the samples $x(n)$ using the sinc interpolation (Use $\Delta t = 0.001$) and superimpose $x(n)$ on it.
3. Reconstruct the analog signal $y_a(t)$ from the samples $x(n)$ using the cubic spline interpolation and superimpose $x(n)$ on it.
4. You should observe that the resultant reconstruction in each case has the correct frequency but a different amplitude. Explain this observation. Comment on the role of phase of $x_a(t)$ on the sampling and reconstruction of signals.

CHAPTER 4

The z-Transform

In Chapter 3 we studied the discrete-time Fourier transform approach for representing discrete signals using complex exponential sequences. This representation clearly has advantages for LTI systems because it describes systems in the frequency domain using the frequency response function $H(e^{j\omega})$. The computation of the sinusoidal steady-state response is greatly facilitated by the use of $H(e^{j\omega})$. Furthermore, response to any arbitrary absolutely summable sequence $x(n)$ can easily be computed in the frequency domain by multiplying the transform $X(e^{j\omega})$ and the frequency response $H(e^{j\omega})$. However, there are *two* shortcomings to the Fourier transform approach. First, there are many useful signals in practice—such as $u(n)$ and $nu(n)$—for which the discrete-time Fourier transform does not exist. Second, the transient response of a system due to initial conditions or due to changing inputs cannot be computed using the discrete-time Fourier transform approach.

Therefore we now consider an extension of the discrete-time Fourier transform to address these two problems. This extension is called the *z-transform*. Its bilateral (or two-sided) version provides another domain in which a larger class of sequences and systems can be analyzed, and its unilateral (or one-sided) version can be used to obtain system responses with initial conditions or changing inputs.

4.1 THE BILATERAL z-TRANSFORM

The z-transform of a sequence $x(n)$ is given by

$$X(z) \stackrel{\triangle}{=} \mathcal{Z}[x(n)] = \sum_{n=-\infty}^{\infty} x(n)z^{-n} \qquad \textbf{(4.1)}$$

where z is a complex variable. The set of z values for which $X(z)$ exists is called the *region of convergence* (ROC) and is given by

$$R_{x-} < |z| < R_{x+} \tag{4.2}$$

for some non-negative numbers R_{x-} and R_{x+}.

The inverse z-transform of a complex function $X(z)$ is given by

$$x(n) \triangleq \mathcal{Z}^{-1}[X(z)] = \frac{1}{2\pi j}\oint_C X(z)z^{n-1}dz \tag{4.3}$$

where C is a **counterclockwise contour** encircling the origin and lying in the ROC.

Comments:

1. The complex variable z is called the *complex frequency* given by $z = |z|e^{j\omega}$, where $|z|$ is the magnitude and ω is the real frequency.
2. Since the ROC (4.2) is defined in terms of the magnitude $|z|$, the shape of the ROC is an open ring, as shown in Figure 4.1. Note that R_{x-} may be equal to zero and/or R_{x+} could possibly be ∞.
3. If $R_{x+} < R_{x-}$, then the ROC is a *null space* and the z-transform *does not exist.*
4. The function $|z| = 1$ (or $z = e^{j\omega}$) is a circle of unit radius in the z-plane and is called the *unit circle.* If the ROC contains the unit circle, then we can evaluate $X(z)$ on the unit circle.

$$X(z)|_{z=e^{j\omega}} = X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega} = \mathcal{F}[x(n)]$$

Therefore the discrete-time Fourier transform $X(e^{j\omega})$ may be viewed as a special case of the z-transform $X(z)$.

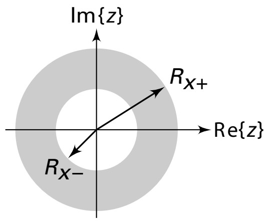


FIGURE 4.1 *A general region of convergence*

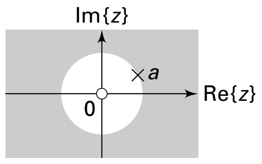


FIGURE 4.2 *The ROC in Example 4.1*

□ **EXAMPLE 4.1** Let $x_1(n) = a^n u(n), \quad 0 < |a| < \infty$. (This sequence is called a *positive-time* sequence). Then

$$X_1(z) = \sum_0^\infty a^n z^{-n} = \sum_0^\infty \left(\frac{a}{z}\right)^n = \frac{1}{1-az^{-1}}; \quad \text{if } \left|\frac{a}{z}\right| < 1$$

$$= \frac{z}{z-a}, \quad |z| > |a| \Rightarrow \text{ROC}_1\text{:} \ \underbrace{|a|}_{R_{x-}} < |z| < \underbrace{\infty}_{R_{x+}}$$

Note: $X_1(z)$ in this example is a rational function; that is,

$$X_1(z) \triangleq \frac{B(z)}{A(z)} = \frac{z}{z-a}$$

where $B(z) = z$ is the *numerator polynomial* and $A(z) = z - a$ is the *denominator polynomial*. The roots of $B(z)$ are called the *zeros* of $X(z)$, whereas the roots of $A(z)$ are called the *poles* of $X(z)$. In this example $X_1(z)$ has a zero at the origin $z = 0$ and a pole at $z = a$. Hence $x_1(n)$ can also be represented by a *pole-zero diagram* in the z-plane in which zeros are denoted by ∘ and poles by × as shown in Figure 4.2. □

□ **EXAMPLE 4.2** Let $x_2(n) = -b^n u(-n-1), 0 < |b| < \infty$. (This sequence is called a *negative-time sequence.*) Then

$$X_2(z) = -\sum_{-\infty}^{-1} b^n z^{-n} = -\sum_{-\infty}^{-1} \left(\frac{b}{z}\right)^n = -\sum_1^\infty \left(\frac{z}{b}\right)^n = 1 - \sum_0^\infty \left(\frac{z}{b}\right)^n$$

$$= 1 - \frac{1}{1 - z/b} = \frac{z}{z-b}, \quad \text{ROC}_2\text{:} \ \underbrace{0}_{R_{x-}} < |z| < \underbrace{|b|}_{R_{x+}}$$

The ROC$_2$ and the pole-zero plot for this $x_2(n)$ are shown in Figure 4.3.

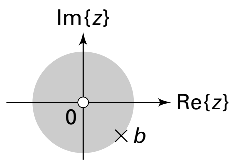


FIGURE 4.3 *The ROC in Example 4.2*

Note: If $b = a$ in this example, then $X_2(z) = X_1(z)$ except for their respective ROCs; that is, $\text{ROC}_1 \neq \text{ROC}_2$. This implies that the ROC is a distinguishing feature that guarantees the uniqueness of the z-transform. Hence it plays a very important role in system analysis. □

□ **EXAMPLE 4.3** Let $x_3(n) = x_1(n) + x_2(n) = a^n u(n) - b^n u(-n-1)$ (This sequence is called a *two-sided sequence.*) Then using the preceding two examples,

$$\begin{aligned} X_3(z) &= \sum_{n=0}^{\infty} a^n z^{-n} - \sum_{-\infty}^{-1} b^n z^{-n} \\ &= \left\{\frac{z}{z-a}, \text{ROC}_1\text{: } |z| > |a|\right\} + \left\{\frac{z}{z-b}, \text{ROC}_1\text{: } |z| < |b|\right\} \\ &= \frac{z}{z-a} + \frac{z}{z-b}; \quad \text{ROC}_3\text{: } \text{ROC}_1 \cap \text{ROC}_2 \end{aligned}$$

If $|b| < |a|$, than ROC_3 is a null space, and $X_3(z)$ does not exist. If $|a| < |b|$, then the ROC_3 is $|a| < |z| < |b|$, and $X_3(z)$ exists in this region as shown in Figure 4.4. □

4.1.1 PROPERTIES OF THE ROC

From the observation of the ROCs in the preceding three examples, we state the following properties.

1. The ROC is **always bounded by a circle** since the convergence condition is on the magnitude $|z|$.
2. The sequence $x_1(n) = a^n u(n)$ in Example 4.1 is a special case of a *right-sided sequence*, defined as a sequence $x(n)$ that is zero for some $n < n_0$. From Example 4.1, the ROC for right-sided sequences is **always outside of a circle of radius R_{x-}**. If $n_0 \geq 0$, then the right-sided sequence is also called a *causal* sequence.
3. The sequence $x_2(n) = -b^n u(-n-1)$ in Example 4.2 is a special case of a *left-sided* sequence, defined as a sequence $x(n)$ that is zero for some $n > n_0$. If $n_0 \leq 0$, the resulting sequence is called an *anticausal* sequence. From Example 4.2, the ROC for left-sided sequences is **always inside of a circle of radius R_{x+}**.

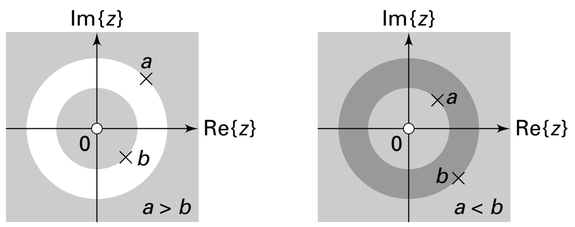


FIGURE 4.4 *The ROC in Example 4.3*

4. The sequence $x_3(n)$ in Example 4.3 is a two-sided sequence. The ROC for two-sided sequences is **always an open ring $R_{x-} < |z| < R_{x+}$**, if it exists.
5. The sequences that are zero for $n < n_1$ and $n > n_2$ are called *finite-duration sequences*. The ROC for such sequences is **the entire z-plane**. If $n_1 < 0$, then $z = \infty$ is not in the ROC. If $n_2 > 0$, then $z = 0$ is not in the ROC.
6. The ROC cannot include a pole since $X(z)$ converges uniformly in there.
7. There is at least one pole on the boundary of a ROC of a rational $X(z)$.
8. The ROC is one contiguous region; that is, the ROC does not come in pieces.

In digital signal processing, signals are assumed to be causal since almost every digital data is acquired in real time. Therefore the only ROC of interest to us is the one given in statement 2.

4.2 IMPORTANT PROPERTIES OF THE z-TRANSFORM

The properties of the z-transform are generalizations of the properties of the discrete-time Fourier transform that we studied in Chapter 3. We state the following important properties of the z-transform without proof.

1. **Linearity:**

$$\mathcal{Z}\left[a_1x_1(n) + a_2x_2(n)\right] = a_1X_1(z) + a_2X_2(z); \quad \text{ROC: } \text{ROC}_{x_1} \cap \text{ROC}_{x_2} \tag{4.4}$$

2. **Sample shifting:**

$$\mathcal{Z}\left[x\left(n - n_0\right)\right] = z^{-n_0}X(z); \quad \text{ROC: ROC}_x \tag{4.5}$$

3. **Frequency shifting:**

$$\mathcal{Z}\left[a^n x(n)\right] = X\left(\frac{z}{a}\right); \quad \text{ROC: ROC}_x \text{ scaled by } |a| \tag{4.6}$$

4. **Folding:**

$$\mathcal{Z}\left[x\left(-n\right)\right] = X\left(1/z\right); \quad \text{ROC: Inverted ROC}_x \tag{4.7}$$

5. **Complex conjugation:**

$$\mathcal{Z}\left[x^*(n)\right] = X^*(z^*); \quad \text{ROC: ROC}_x \tag{4.8}$$

6. **Differentiation in the z-domain:**

$$\mathcal{Z}\left[nx(n)\right] = -z\frac{dX(z)}{dz}; \quad \text{ROC: ROC}_x \tag{4.9}$$

   This property is also called the *multiplication-by-a-ramp property.*

7. **Multiplication:**

$$\mathcal{Z}\left[x_1(n)x_2\left(n\right)\right] = \frac{1}{2\pi j}\oint_C X_1(\nu)X_2\left(z/\nu\right)\nu^{-1}d\nu; \quad \text{ROC: ROC}_{x_1} \cap \text{Inverted ROC}_{x_2} \tag{4.10}$$

   where C is a closed contour that encloses the origin and lies in the common ROC.

8. **Convolution:**

$$\mathcal{Z}\left[x_1(n) * x_2(n)\right] = X_1(z)X_2(z); \quad \text{ROC: ROC}_{x_1} \cap \text{ROC}_{x_2} \tag{4.11}$$

This last property transforms the time-domain convolution operation into a multiplication between two functions. It is a significant property in many ways. First, if $X_1(z)$ and $X_2(z)$ are two polynomials, then their product can be implemented using the `conv` function in MATLAB.

□ **EXAMPLE 4.4** Let $X_1(z) = 2 + 3z^{-1} + 4z^{-2}$ and $X_2(z) = 3 + 4z^{-1} + 5z^{-2} + 6z^{-3}$. Determine $X_3(z) = X_1(z)X_2(z)$.

Solution From the definition of the z-transform, we observe that

$$x_1(n) = \{\underset{\uparrow}{2}, 3, 4\} \qquad \text{and} \qquad x_2(n) = \{\underset{\uparrow}{3}, 4, 5, 6\}$$

Then the convolution of these two sequences will give the coefficients of the required polynomial product.

MATLAB script:

```
>> x1 = [2,3,4]; x2 = [3,4,5,6]; x3 = conv(x1,x2)
x3 =         6     17     34     43     38     24
```

Hence

$$X_3(z) = 6 + 17z^{-1} + 34z^{-2} + 43z^{-3} + 38z^{-4} + 24z^{-5}$$

Using the `conv_m` function developed in Chapter 2, we can also multiply two z-domain polynomials corresponding to noncausal sequences. □

□ **EXAMPLE 4.5** Let $X_1(z) = z + 2 + 3z^{-1}$ and $X_2(z) = 2z^2 + 4z + 3 + 5z^{-1}$. Determine $X_3(z) = X_1(z)X_2(z)$.

Solution Note that

$$x_1(n) = \{1, \underset{\uparrow}{2}, 3\} \qquad \text{and} \qquad x_2(n) = \{2, 4, \underset{\uparrow}{3}, 5\}$$

Using the MATLAB script,

```
>> x1 = [1,2,3]; n1 = [-1:1];  x2 = [2,4,3,5]; n2 = [-2:1];
>> [x3,n3] = conv_m(x1,n1,x2,n2)
x3 =
     2     8    17    23    19   15
n3 =
    -3    -2    -1     0     1     2
```

we have

$$X_3(z) = 2z^3 + 8z^2 + 17z + 23 + 19z^{-1} + 15z^{-2} \qquad \square$$

In passing we note that to divide one polynomial by another one, we would require an inverse operation called *deconvolution* [23, Chapter 6]. In MATLAB `[p,r] = deconv(b,a)` computes the result of dividing `b` by `a` in a polynomial part `p` and a remainder `r`. For example, if we divide the polynomial $X_3(z)$ in Example 4.4 by $X_1(z)$, as follows,

```
>> x3 = [6,17,34,43,38,24]; x1 = [2,3,4]; [x2,r] = deconv(x3,x1)
x2 =
     3     4     5     6
r =
     0     0     0     0     0     0
```

then we obtain the coefficients of the polynomial $X_2(z)$ as expected. To obtain the sample index, we will have to modify the `deconv` function as we did in the `conv_m` function. This is explored in Problem P4.10. This operation is useful in obtaining a *proper* rational part from an *improper* rational function.

The second important use of the convolution property is in system output computations as we shall see in a later section. This interpretation is particularly useful for verifying the z-transform expression $X(z)$ of a casual sequence using MATLAB. Note that since MATLAB is a numerical processor (unless the Symbolic toolbox is used), it cannot be used for symbolic z-transform calculations. We will now elaborate on this. Let $x(n)$ be a sequence with a rational transform

$$X(z) = \frac{B(z)}{A(z)}$$

where $B(z)$ and $A(z)$ are polynomials in z^{-1}. If we use the coefficients of $B(z)$ and $A(z)$ as the `b` and `a` arrays in the `filter` routine and excite this filter by the impulse sequence $\delta(n)$, then from (4.11) and using $\mathcal{Z}[\delta(n)] = 1$, the output of the filter will be $x(n)$. (This is a numerical approach of computing the inverse z-transform; we will discuss the analytical approach in the next section.) We can compare this output with the given $x(n)$ to verify that $X(z)$ is indeed the transform of $x(n)$. This is illustrated in Example 4.6. An equivalent approach is to use the `impz` function discussed in Chapter 2.

4.2.1 SOME COMMON z-TRANSFORM PAIRS

Using the definition of z-transform and its properties, one can determine z-transforms of common sequences. A list of some of these sequences is given in Table 4.1.

TABLE 4.1 *Some common z-transform pairs*

Sequence	*Transform*	*ROC*
$\delta(n)$	1	$\forall z$
$u(n)$	$\dfrac{1}{1-z^{-1}}$	$\|z\| > 1$
$-u(-n-1)$	$\dfrac{1}{1-z^{-1}}$	$\|z\| < 1$
$a^n u(n)$	$\dfrac{1}{1-az^{-1}}$	$\|z\| > \|a\|$
$-b^n u(-n-1)$	$\dfrac{1}{1-bz^{-1}}$	$\|z\| < \|b\|$
$[a^n \sin\omega_0 n]\, u(n)$	$\dfrac{(a\sin\omega_0)z^{-1}}{1-(2a\cos\omega_0)z^{-1}+a^2z^{-2}}$	$\|z\| > \|a\|$
$[a^n \cos\omega_0 n]\, u(n)$	$\dfrac{1-(a\cos\omega_0)z^{-1}}{1-(2a\cos\omega_0)z^{-1}+a^2z^{-2}}$	$\|z\| > \|a\|$
$na^n u(n)$	$\dfrac{az^{-1}}{(1-az^{-1})^2}$	$\|z\| > \|a\|$
$-nb^n u(-n-1)$	$\dfrac{bz^{-1}}{(1-bz^{-1})^2}$	$\|z\| < \|b\|$

□ **EXAMPLE 4.6** Using z-transform properties and the z-transform table, determine the z-transform of

$$x(n) = (n-2)(0.5)^{(n-2)} \cos\left[\frac{\pi}{3}(n-2)\right] u(n-2)$$

Solution Applying the sample-shift property,

$$X(z) = \mathcal{Z}[x(n)] = z^{-2}\mathcal{Z}\left[n(0.5)^n \cos\left(\frac{\pi n}{3}\right) u(n)\right]$$

with no change in the ROC. Applying the multiplication by a ramp property,

$$X(z) = z^{-2}\left\{-z\frac{d\mathcal{Z}[(0.5)^n\cos(\frac{\pi}{3}n)u(n)]}{dz}\right\}$$

with no change in the ROC. Now the z-transform of $(0.5)^n\cos(\frac{\pi}{3}n)u(n)$ from Table 4.1 is

$$\begin{aligned}\mathcal{Z}\left[(0.5)^n\cos\left(\frac{\pi n}{3}\right)u(n)\right] &= \frac{1-(0.5\cos\frac{\pi}{3})z^{-1}}{1-2(0.5\cos\frac{\pi}{3})z^{-1}+0.25z^{-2}}; \quad |z|>0.5\\ &= \frac{1-0.25z^{-1}}{1-0.5z^{-1}+0.25z^{-2}}; \quad |z|>0.5\end{aligned}$$

Hence

$$\begin{aligned}X(z) &= -z^{-1}\frac{d}{dz}\left\{\frac{1-0.25z^{-1}}{1-0.5z^{-1}+0.25z^{-2}}\right\}, && |z|>0.5\\ &= -z^{-1}\left\{\frac{-0.25z^{-2}+0.5z^{-3}-0.0625z^{-4}}{1-z^{-1}+0.75z^{-2}-0.25z^{-3}+0.0625z^{-4}}\right\}, && |z|>0.5\\ &= \frac{0.25z^{-3}-0.5z^{-4}+0.0625z^{-5}}{1-z^{-1}+0.75z^{-2}-0.25z^{-3}+0.0625z^{-4}}, && |z|>0.5\end{aligned}$$

MATLAB verification: To check that this $X(z)$ is indeed the correct expression, let us compute the first 8 samples of the sequence $x(n)$ corresponding to $X(z)$, as discussed before.

```
>> b = [0,0,0,0.25,-0.5,0.0625]; a = [1,-1,0.75,-0.25,0.0625];
>> [delta,n]=impseq(0,0,7)
delta =
     1     0     0     0     0     0     0     0
n =
     0     1     2    3     4     5     6     7
>> x = filter(b,a,delta) % check sequence
x =
  Columns 1 through 4
                    0                 0                  0   0.25000000000000
  Columns 5 through 8
 -0.25000000000000  -0.37500000000000  -0.12500000000000   0.07812500000000
>> x = [(n-2).*(1/2).^(n-2).*cos(pi*(n-2)/3)].*stepseq(2,0,7) % original sequence
x =
 Columns 1 through 4
                   0                 0                  0   0.25000000000000
  Columns 5 through 8
  -0.25000000000000  -0.37500000000000  -0.12500000000000   0.07812500000000
```

This approach can be used to verify the z-transform computations. □

4.3 INVERSION OF THE z-TRANSFORM

From equation (4.3), the inverse z-transform computation requires an evaluation of a complex contour integral that, in general, is a complicated procedure. The most practical approach is to use the partial fraction expansion method. It makes use of the z-transform Table 4.1 (or similar tables available in many textbooks). The z-transform, however, must be a rational function. This requirement is generally satisfied in digital signal processing.

Central Idea

- When $X(z)$ is a rational function of z^{-1}, it can be expressed as a sum of simple factors using the partial fraction expansion. The individual sequences corresponding to these factors can then be written down using the z-transform table.
  The inverse z-transform procedure can be summarized as follows:

Method

- Given

$$X(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}, \; R_{x-} < |z| < R_{x+} \tag{4.12}$$

- express it as

$$X(z) = \underbrace{\frac{\tilde{b}_0 + \tilde{b}_1 z^{-1} + \cdots + \tilde{b}_{N-1} z^{-(N-1)}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}}}_{\text{Proper rational part}} + \underbrace{\sum_{k=0}^{M-N} C_k z^{-k}}_{\text{polynomial part if } M \geq N}$$

  where the first term on the right-hand side is the proper rational part, and the second term is the polynomial (finite-length) part. This can be obtained by performing polynomial division if $M \geq N$ using the `deconv` function.
- Perform a partial fraction expansion on the proper rational part of $X(z)$ to obtain

$$X(z) = \sum_{k=1}^{N} \frac{R_k}{1 - p_k z^{-1}} + \underbrace{\sum_{k=0}^{M-N} C_k z^{-k}}_{M \geq N} \tag{4.13}$$

where p_k is the kth pole of $X(z)$ and R_k is the residue at p_k. It is assumed that the poles are distinct for which the residues are given by

$$R_k = \frac{\tilde{b}_0 + \tilde{b}_1 z^{-1} + \cdots + \tilde{b}_{N-1} z^{-(N-1)}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} (1 - p_k z^{-1}) \Bigg|_{z=p_k}$$

For repeated poles the expansion (4.13) has a more general form. If a pole p_k has multiplicity r, then its expansion is given by

$$\sum_{\ell=1}^{r} \frac{R_{k,\ell} z^{-(\ell-1)}}{(1 - p_k z^{-1})^{\ell}} = \frac{R_{k,1}}{1 - p_k z^{-1}} + \frac{R_{k,2} z^{-1}}{(1 - p_k z^{-1})^2} + \cdots + \frac{R_{k,r} z^{-(r-1)}}{(1 - p_k z^{-1})^r} \tag{4.14}$$

where the residues $R_{k,\ell}$ are computed using a more general formula, which is available in reference [23].

- assuming distinct poles as in (4.13), write $x(n)$ as

$$x(n) = \sum_{k=1}^{N} R_k \mathcal{Z}^{-1}\left[\frac{1}{1 - p_k z^{-1}}\right] + \underbrace{\sum_{k=0}^{M-N} C_k \delta(n-k)}_{M \geq N}$$

- finally, use the relation from Table 4.1

$$\mathcal{Z}^{-1}\left[\frac{z}{z - p_k}\right] = \begin{cases} p_k^n u(n) & |z_k| \leq R_{x-} \\ -p_k^n u(-n-1) & |z_k| \geq R_{x+} \end{cases} \tag{4.15}$$

to complete $x(n)$.

A similar procedure is used for repeated poles.

☐ **EXAMPLE 4.7** Find the inverse z-transform of $x(z) = \dfrac{z}{3z^2 - 4z + 1}$.

Solution Write

$$\begin{aligned} X(z) &= \frac{z}{3(z^2 - \frac{4}{3}z + \frac{1}{3})} = \frac{\frac{1}{3}z^{-1}}{1 - \frac{4}{3}z^{-1} + \frac{1}{3}z^{-2}} \\ &= \frac{\frac{1}{3}z^{-1}}{(1 - z^{-1})(1 - \frac{1}{3}z^{-1})} = \frac{\frac{1}{2}}{1 - z^{-1}} - \frac{\frac{1}{2}}{1 - \frac{1}{3}z^{-1}} \end{aligned}$$

or

$$X(z) = \frac{1}{2}\left(\frac{1}{1 - z^{-1}}\right) - \frac{1}{2}\left(\frac{1}{1 - \frac{1}{3}z^{-1}}\right)$$

Now, $X(z)$ has two poles: $z_1 = 1$ and $z_2 = \frac{1}{3}$; and since the ROC is not specified, there are *three* possible ROCs as shown in Figure 4.5.

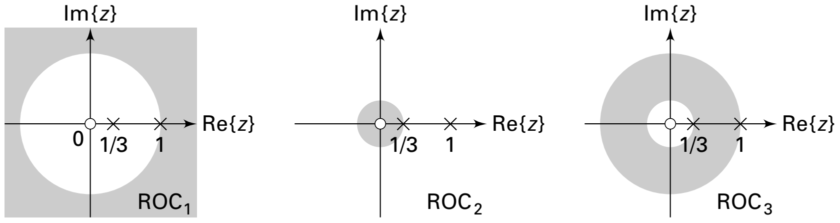


FIGURE 4.5 *The ROCs in Example 4.7*

a. ROC$_1$: $1 < |z| < \infty$. Here both poles are on the interior side of the ROC$_1$; that is, $|z_1| \le R_{x-} = 1$ and $|z_2| \le 1$. Hence from (4.15)

$$x_1(n) = \frac{1}{2}u(n) - \frac{1}{2}\left(\frac{1}{3}\right)^n u(n)$$

which is a right-sided sequence.

b. ROC$_2$: $0 < |z| < \frac{1}{3}$. Here both poles are on the exterior side of the ROC$_2$; that is, $|z_1| \ge R_{x+} = \frac{1}{3}$ and $|z_2| \ge \frac{1}{3}$. Hence from (4.15)

$$\begin{aligned} x_2(n) &= \frac{1}{2}\{-u(-n-1)\} - \frac{1}{2}\left\{-\left(\tfrac{1}{3}\right)^n u(-n-1)\right\} \\ &= \frac{1}{2}\left(\frac{1}{3}\right)^n u(-n-1) - \frac{1}{2}u(-n-1) \end{aligned}$$

which is a left-sided sequence.

c. ROC$_3$: $\frac{1}{3} < |z| < 1$. Here pole z_1 is on the exterior side of the ROC$_3$—that is, $|z_1| \ge R_{x+} = 1$—while pole z_2 is on the interior side—that is, $|z_2| \le \frac{1}{3}$. Hence from (4.15)

$$x_3(n) = -\frac{1}{2}u(-n-1) - \frac{1}{2}\left(\frac{1}{3}\right)^n u(n)$$

which is a two-sided sequence. □

4.3.1 MATLAB IMPLEMENTATION

A MATLAB function `residuez` is available to compute the residue part and the direct (or polynomial) terms of a rational function in z^{-1}. Let

$$\begin{aligned} X(z) &= \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \cdots + a_N z^{-N}} = \frac{B(z)}{A(z)} \\ &= \sum_{k=1}^{N} \frac{R_k}{1 - p_k z^{-1}} + \underbrace{\sum_{k=0}^{M-N} C_k z^{-k}}_{M \ge N} \end{aligned}$$

be a rational function in which the numerator and the denominator polynomials are in *ascending* powers of z^{-1}. Then `[R,p,C]=residuez(b,a)` computes the residues, poles, and direct terms of $X(z)$ in which two polynomials $B(z)$ and $A(z)$ are given in two vectors `b` and `a`, respectively. The returned column vector `R` contains the residues, column vector `p` contains the pole locations, and row vector `C` contains the direct terms. If `p(k)=...=p(k+r-1)` is a pole of multiplicity `r`, then the expansion includes the term of the form

$$\frac{R_k}{1-p_kz^{-1}}+\frac{R_{k+1}}{(1-p_kz^{-1})^2}+\cdots+\frac{R_{k+r-1}}{(1-p_kz^{-1})^r} \tag{4.16}$$

which is different from (4.14).

Similarly, `[b,a]=residuez(R,p,C)`, with three input arguments and two output arguments, converts the partial fraction expansion back to polynomials with coefficients in row vectors `b` and `a`.

□ **EXAMPLE 4.8** To check our residue calculations, let us consider the rational function

$$X(z)=\frac{z}{3z^2-4z+1}$$

given in Example 4.7.

Solution First rearrange $X(z)$ so that it is a function in ascending powers of z^{-1}.

$$X(z)=\frac{z^{-1}}{3-4z^{-1}+z^{-2}}=\frac{0+z^{-1}}{3-4z^{-1}+z^{-2}}$$

Now using the MATLAB script

```
>> b = [0,1]; a = [3,-4,1]; [R,p,C] = residuez(b,a)
R =
    0.5000
   -0.5000
p =
    1.0000
    0.3333
c =
      []
```

we obtain

$$X(z)=\frac{\frac{1}{2}}{1-z^{-1}}-\frac{\frac{1}{2}}{1-\frac{1}{3}z^{-1}}$$

as before. Similarly, to convert back to the rational function form,

```
>> [b,a] = residuez(R,p,C)
b =
    0.0000
    0.3333
```

```
a =
     1.0000
    -1.3333
     0.3333
```

so that

$$X(z) = \frac{0+\frac{1}{3}z^{-1}}{1-\frac{4}{3}z^{-1}+\frac{1}{3}z^{-2}} = \frac{z^{-1}}{3-4z^{-1}+z^{-2}} = \frac{z}{3z^2-4z+1}$$

as before. □

□ **EXAMPLE 4.9** Compute the inverse z-transform of

$$X(z) = \frac{1}{\left(1-0.9z^{-1}\right)^2\left(1+0.9z^{-1}\right)}, \quad |z| > 0.9$$

Solution We will evaluate the denominator polynomial as well as the residues using the MATLAB script:

```
>> b = 1; a = poly([0.9,0.9,-0.9])
a =
      1.0000    -0.9000    -0.8100     0.7290
>> [R,p,C]=residuez(b,a)
R =
     0.2500
     0.5000
     0.2500
p =
     0.9000
     0.9000
    -0.9000
c =
     []
```

Note that the denominator polynomial is computed using MATLAB's polynomial function `poly`, which computes the polynomial coefficients, given its roots. We could have used the `conv` function, but the use of the `poly` function is more convenient for this purpose. From the residue calculations and using the order of residues given in (4.16), we have

$$X(z) = \frac{0.25}{1-0.9z^{-1}} + \frac{0.5}{\left(1-0.9z^{-1}\right)^2} + \frac{0.25}{1+0.9z^{-1}}, \qquad |z| > 0.9$$

$$= \frac{0.25}{1-0.9z^{-1}} + \frac{0.5}{0.9}z\frac{\left(0.9z^{-1}\right)}{\left(1-0.9z^{-1}\right)^2} + \frac{0.25}{1+0.9z^{-1}}, \quad |z| > 0.9$$

Hence from Table 4.1 and using the z-transform property of time-shift,

$$x(n) = 0.25(0.9)^n u(n) + \frac{5}{9}(n+1)(0.9)^{n+1}u(n+1) + 0.25\left(-0.9\right)^n u(n)$$

which, upon simplification, becomes

$$x(n) = 0.75(0.9)^n u(n) + 0.5n(0.9)^n u(n) + 0.25\left(-0.9\right)^n u(n)$$

MATLAB verification:

```
>> [delta,n] = impseq(0,0,7);  x = filter(b,a,delta) % check sequence
x =
  Columns 1 through 4
   1.00000000000000   0.90000000000000   1.62000000000000   1.45800000000000
  Columns 5 through 8
   1.96830000000000   1.77147000000000   2.12576400000000   1.91318760000000
>> x = (0.75)*(0.9).^n + (0.5)*n.*(0.9).^n + (0.25)*(-0.9).^n % answer sequence
x =
  Columns 1 through 4
   1.00000000000000   0.90000000000000   1.62000000000000   1.45800000000000
  Columns 5 through 8
   1.96830000000000   1.77147000000000   2.12576400000000   1.91318760000000
```

□

□ **EXAMPLE 4.10** Determine the inverse z-transform of

$$X(z) = \frac{1 + 0.4\sqrt{2}z^{-1}}{1 - 0.8\sqrt{2}z^{-1} + 0.64z^{-2}}$$

so that the resulting sequence is causal and contains no complex numbers.

Solution We will have to find the poles of $X(z)$ in the polar form to determine the ROC of the causal sequence.

MATLAB script:

```
>> b = [1,0.4*sqrt(2)]; a=[1,-0.8*sqrt(2),0.64];
>> [R,p,C] = residuez(b,a)
R =
   0.5000 - 1.0000i
   0.5000 + 1.0000i
p =
   0.5657 + 0.5657i
   0.5657 - 0.5657i
C =
     []
>> Mp=(abs(p))'   % pole magnitudes
Mp =
    0.8000    0.8000
>> Ap=(angle(p))'/pi   % pole angles in pi units
Ap =
   0.2500   -0.2500
```

From these calculations

$$X(z) = \frac{0.5-j}{1-0.8e^{+j\frac{\pi}{4}}z^{-1}} + \frac{0.5+j}{1-0.8e^{-j\frac{\pi}{4}}z^{-1}}, \quad |z| > 0.8$$

and from Table 4.1, we have

$$\begin{aligned} x(n) &= (0.5-j)\,0.8^n e^{+j\frac{\pi}{4}n}u(n) + (0.5+j)\,0.8^n e^{-j\frac{\pi}{4}n}u(n) \\ &= 0.8^n[0.5\{e^{+j\frac{\pi}{4}n} + e^{-j\frac{\pi}{4}n}\} - j\{e^{+j\frac{\pi}{4}n} - e^{-j\frac{\pi}{4}n}\}]u(n) \\ &= 0.8^n\left[\cos\left(\frac{\pi n}{4}\right) + 2\sin\left(\frac{\pi n}{4}\right)\right]u(n) \end{aligned}$$

MATLAB verification:

```
>> [delta, n] = impseq(0,0,6);
 x = filter(b,a,delta) % check sequence
x =
  Columns 1 through 4
   1.00000000000000   1.69705627484771   1.28000000000000   0.36203867196751
  Columns 5 through 8
  -0.40960000000000  -0.69511425017762  -0.52428800000000  -0.14829104003789
>> x = ((0.8).^n).*(cos(pi*n/4)+2*sin(pi*n/4))
x =
  Columns 1 through 4
   1.00000000000000   1.69705627484771   1.28000000000000   0.36203867196751
  Columns 5 through 8
  -0.40960000000000  -0.69511425017762  -0.52428800000000  -0.14829104003789
```
□

4.4 SYSTEM REPRESENTATION IN THE z-DOMAIN

Similar to the frequency response function $H(e^{j\omega})$, we can define the z-domain function, $H(z)$, called the *system function*. However, unlike $H(e^{j\omega})$, $H(z)$ exists for systems that may not be BIBO stable.

■ **DEFINITION 1** [The System Function] The system function $H(z)$ is given by

$$H(z) \triangleq \mathcal{Z}[h(n)] = \sum_{-\infty}^{\infty} h(n)z^{-n}; \quad R_{h-} < |z| < R_{h+} \tag{4.17}$$

Using the convolution property (4.11) of the z-transform, the output transform $Y(z)$ is given by

$$Y(z) = H(z)\,X(z) \quad : \ \mathrm{ROC}_y = \mathrm{ROC}_h \cap \mathrm{ROC}_x \tag{4.18}$$

provided ROC_x overlaps with ROC_h. Therefore a linear and time-invariant system can be represented in the z-domain by

$$X(z) \longrightarrow \boxed{H(z)} \longrightarrow Y(z) = H(z)\, X(z)$$

4.4.1 SYSTEM FUNCTION FROM THE DIFFERENCE EQUATION REPRESENTATION

When LTI systems are described by a difference equation

$$y(n) + \sum_{k=1}^{N} a_k y(n-k) = \sum_{\ell=0}^{M} b_\ell x(n-\ell) \tag{4.19}$$

the system function $H(z)$ can easily be computed. Taking the z-transform of both sides, and using properties of the z-transform,

$$Y(z) + \sum_{k=1}^{N} a_k z^{-k} Y(z) = \sum_{\ell=0}^{M} b_\ell z^{-\ell} X(z)$$

or

$$\begin{aligned} H(z) \stackrel{\triangle}{=} \frac{Y(z)}{X(z)} &= \frac{\sum_{\ell=0}^{M} b_\ell z^{-\ell}}{1 + \sum_{k=1}^{N} a_k z^{-k}} = \frac{B(z)}{A(z)} \\ &= \frac{b_0 z^{-M}\left(z^M + \cdots + \frac{b_M}{b_0}\right)}{z^{-N}\left(z^N + \cdots + a_N\right)} \end{aligned} \tag{4.20}$$

After factorization, we obtain

$$H(z) = b_0\, z^{N-M}\, \frac{\prod_{\ell=1}^{N}(z - z_\ell)}{\prod_{k=1}^{N}(z - p_k)} \tag{4.21}$$

where z_ℓs are the system zeros and p_k's are the system poles. Thus $H(z)$ (and hence an LTI system) can also be represented in the z-domain using a pole-zero plot. This fact is useful in designing simple filters by proper placement of poles and zeros.

To determine zeros and poles of a rational $H(z)$, we can use the MATLAB function `roots` on both the numerator and the denominator polynomials. (Its inverse function `poly` determines polynomial coefficients from its roots, as discussed in the previous section.) It is also possible to use MATLAB to plot these roots for a visual display of a pole-zero plot. The function `zplane(b,a)` plots poles and zeros, given the numerator *row* vector `b` and the denominator *row* vector `a`. As before, the symbol `o` represents a zero and the symbol `x` represents a pole. The plot includes the unit circle for reference. Similarly, `zplane(z,p)` plots the zeros in *column* vector `z` and the poles in *column* vector `p`. Note very carefully the form of the input arguments for the proper use of this function.

4.4.2 TRANSFER FUNCTION REPRESENTATION

If the ROC of $H(z)$ includes a unit circle ($z = e^{j\omega}$), then we can evaluate $H(z)$ on the unit circle, resulting in a frequency response function or transfer function $H(e^{j\omega})$. Then from (4.21)

$$H(e^{j\omega}) = b_0\, e^{j(N-M)\omega}\, \frac{\prod_1^M (e^{j\omega} - z_\ell)}{\prod_1^N (e^{j\omega} - p_k)} \tag{4.22}$$

The factor $(e^{j\omega} - z_\ell)$ can be interpreted as a *vector* in the complex z-plane from a zero z_ℓ to the unit circle at $z = e^{j\omega}$, while the factor $(e^{j\omega} - p_k)$ can be interpreted as a vector from a pole p_k to the unit circle at $z = e^{j\omega}$. This is shown in Figure 4.6. Hence the magnitude response function

$$|H(e^{j\omega})| = |b_0| \frac{|e^{j\omega} - z_1| \cdots |e^{j\omega} - z_M|}{|e^{j\omega} - p_1| \cdots |e^{j\omega} - p_N|} \tag{4.23}$$

can be interpreted as a product of the lengths of vectors from zeros to the unit circle *divided* by the lengths of vectors from poles to the unit circle and *scaled* by $|b_0|$. Similarly, the phase response function

$$\angle H(e^{j\omega}) = \underbrace{[0 \text{ or } \pi]}_{\text{Constant}} + \underbrace{[(N-M)\,\omega]}_{\text{Linear}} + \underbrace{\sum_1^M \angle(e^{j\omega} - z_k) - \sum_1^N \angle(e^{j\omega} - p_k)}_{\text{Nonlinear}} \tag{4.24}$$

can be interpreted as a sum of a constant factor, a linear-phase factor, and a nonlinear-phase factor (angles from the "zero vectors" *minus* the sum of angles from the "pole vectors").

4.4.3 MATLAB IMPLEMENTATION

In Chapter 3, we plotted magnitude and phase responses in MATLAB by directly implementing their functional forms. MATLAB also provides

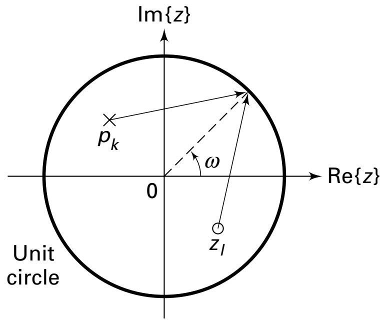


FIGURE 4.6 *Pole and zero vectors*

a function called `freqz` for this computation, which uses the preceding interpretation. In its simplest form, this function is invoked by

```
[H,w] = freqz(b,a,N)
```

which returns the N-point frequency vector `w` and the N-point complex frequency response vector `H` of the system, given its numerator and denominator coefficients in vectors `b` and `a`. The frequency response is evaluated at `N` points equally spaced around the upper half of the unit circle. Note that the `b` and `a` vectors are the same vectors we use in the `filter` function or derived from the difference equation representation (4.19).

The second form

```
[H,w] = freqz(b,a,N,'whole')
```

uses `N` points around the whole unit circle for computation.

In yet another form

```
H = freqz(b,a,w)
```

it returns the frequency response at frequencies designated in vector `w`, normally between 0 and π. It should be noted that the `freqz` function can also be used for numerical computation of the DTFT of a *finite-duration, causal* sequence $x(n)$. In this approach, `b = x` and `a = 1`.

□ **EXAMPLE 4.11** Given a causal system

$$y(n) = 0.9y(n-1) + x(n)$$

a. Determine $H(z)$ and sketch its pole-zero plot.
b. Plot $|H(e^{j\omega})|$ and $\angle H(e^{j\omega})$.
c. Determine the impulse response $h(n)$.

Solution The difference equation can be put in the form

$$y(n) - 0.9y(n-1) = x(n)$$

a. From (4.21)

$$H(z) = \frac{1}{1-0.9z^{-1}}; \quad |z| > 0.9$$

since the system is causal. There is one pole at 0.9 and one zero at the origin. We will use MATLAB to illustrate the use of the `zplane` function.

```
>> b = [1, 0]; a = [1, -0.9];  zplane(b,a)
```

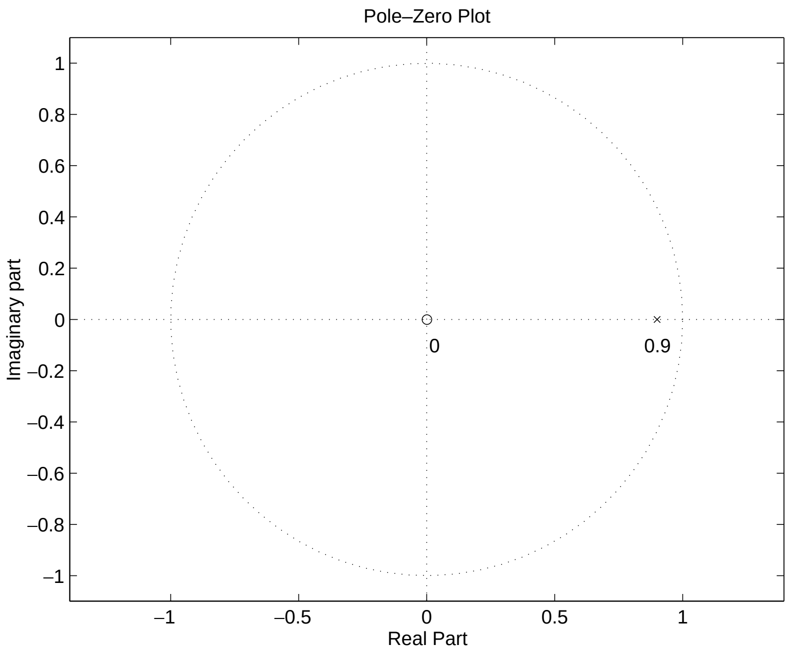


FIGURE 4.7 *Pole-zero plot of Example 4.11a*

Note that we specified `b=[1,0]` instead of `b=1` because the `zplane` function assumes that scalars are zeros or poles. The resulting pole-zero plot is shown in Figure 4.7.

b. Using (4.23) and (4.24), we can determine the magnitude and phase of $H(e^{j\omega})$. Once again we will use MATLAB to illustrate the use of the `freqz` function. Using its first form, we will take 100 points along the upper half of the unit circle.

MATLAB Script:

```
>> [H,w] = freqz(b,a,100);  magH = abs(H); phaH = angle(H);
>> subplot(2,1,1);plot(w/pi,magH);grid
>> xlabel('frequency in pi units'); ylabel('Magnitude');
>> title('Magnitude Response')
>> subplot(2,1,2);plot(w/pi,phaH/pi);grid
>> xlabel('frequency in pi units'); ylabel('Phase in pi units');
>> title('Phase Response')
```

The response plots are shown in Figure 4.8. If you study these plots carefully, you will observe that the plots are computed between $0 \leq \omega \leq 0.99\pi$ and fall short at $\omega = \pi$. This is due to the fact that in MATLAB the lower half

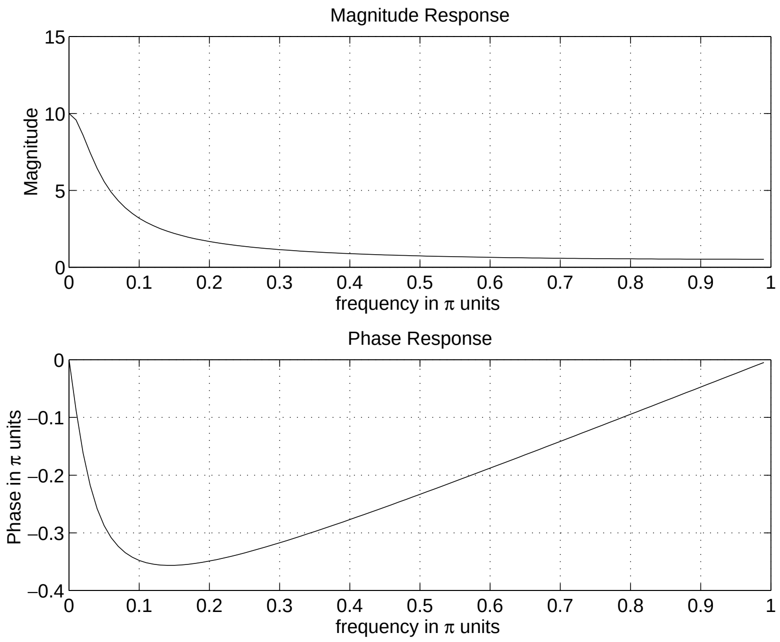


FIGURE 4.8 *Frequency response plots in Example 4.11*

of the unit circle begins at $\omega = \pi$. To overcome this problem, we will use the second form of the `freqz` function as follows.

```
>> [H,w] = freqz(b,a,200,'whole');
>> magH = abs(H(1:101)); phaH = angle(H(1:101));
```

Now the 101st element of the array `H` will correspond to $\omega = \pi$. A similar result can be obtained using the third form of the `freqz` function.

```
>> w = [0:1:100]*pi/100;  H = freqz(b,a,w);
>> magH = abs(H); phaH = angle(H);
```

In the future we will use any one of these forms, depending on our convenience. Also note that in the plots we divided the `w` and `phaH` arrays by `pi` so that the plot axes are in the units of π and easier to read. *This practice is strongly recommended.*

c. From the z-transform in Table 4.1

$$h(n) = \mathcal{Z}^{-1}\left[\frac{1}{1-0.9z^{-1}},\ |z| > 0.9\right] = (0.9)^n u(n) \qquad \square$$

□ **EXAMPLE 4.12** Given that

$$H(z) = \frac{z+1}{z^2 - 0.9z + 0.81}$$

is a causal system, find

a. its transfer function representation,
b. its difference equation representation, and
c. its impulse response representation.

Solution The poles of the system function are at $z = 0.9\angle \pm \pi/3$. Hence the ROC of this causal system is $|z| > 0.9$. Therefore the unit circle is in the ROC, and the discrete-time Fourier transform $H(e^{j\omega})$ exists.

a. Substituting $z = e^{j\omega}$ in $H(z)$,

$$H(e^{j\omega}) = \frac{e^{j\omega}+1}{e^{j2\omega} - 0.9e^{j\omega} + 0.81} = \frac{e^{j\omega}+1}{(e^{j\omega} - 0.9e^{j\pi/3})(e^{j\omega} - 0.9e^{-j\pi/3})}$$

b. Using $H(z) = Y(z)/X(z)$,

$$\frac{Y(z)}{X(z)} = \frac{z+1}{z^2 - 0.9z + 0.81}\left(\frac{z^{-2}}{z^{-2}}\right) = \frac{z^{-1} + z^{-2}}{1 - 0.9z^{-1} + 0.81z^{-2}}$$

Cross multiplying,

$$Y(z) - 0.9z^{-1}Y(z) + 0.81z^{-2}Y(z) = z^{-1}X(z) + z^{-2}X(z)$$

Now taking the inverse z-transform,

$$y(n) - 0.9y(n-1) + 0.81y(n-2) = x(n-1) + x(n-2)$$

or

$$y(n) = 0.9y(n-1) - 0.81y(n-2) + x(n-1) + x(n-2)$$

c. Using the MATLAB script,

```
>> b = [0,1,1]; a = [1,-0.9,0.81];  [R,p,C] = residuez(b,a)
R =
  -0.6173 - 0.9979i
  -0.6173 + 0.9979i
p =
   0.4500 + 0.7794i
   0.4500 - 0.7794i
C =
    1.2346
>> Mp = (abs(p))'
Mp =
    0.9000    0.9000
>> Ap = (angle(p))'/pi
Ap =
   0.3333   -0.3333
```

we have

$$H(z) = 1.2346 + \frac{-0.6173 + j0.9979}{1 - 0.9e^{-j\pi/3}z^{-1}} + \frac{-0.6173 - j0.9979}{1 - 0.9e^{j\pi/3}z^{-1}}, \quad |z| > 0.9$$

Hence from Table 4.1

$$\begin{aligned} h(n) &= 1.2346\delta(n) + [(-0.6173 + j0.9979)0.9^n e^{-j\pi n/3} \\ &\quad +(-0.6173 - j0.9979)0.9^n e^{j\pi n/3}]u(n) \\ &= 1.2346\delta(n) + 0.9^n[-1.2346\cos(\pi n/3) + 1.9958\sin(\pi n/3)]u(n) \\ &= 0.9^n[-1.2346\cos(\pi n/3) + 1.9958\sin(\pi n/3)]u(n-1) \end{aligned}$$

The last step results from the fact that $h(0) = 0$. □

4.4.4 RELATIONSHIPS BETWEEN SYSTEM REPRESENTATIONS

In this and the previous two chapters, we developed several system representations. Figure 4.9 depicts the relationships among these representations in a graphical form.

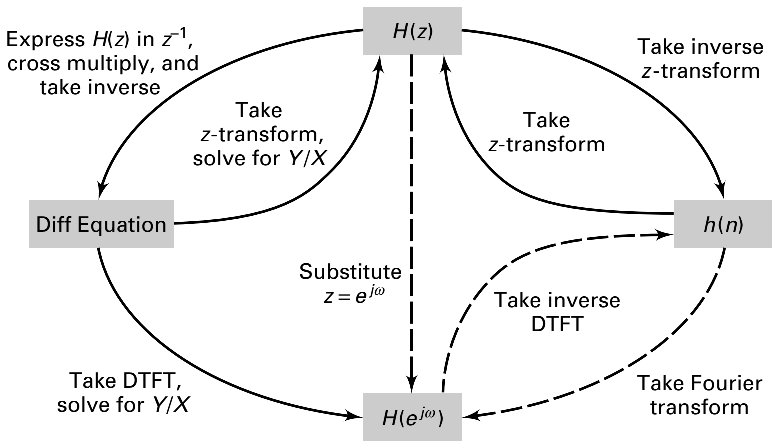


FIGURE 4.9 *System representations in pictorial form*

4.4.5 STABILITY AND CAUSALITY

For LTI systems, the BIBO stability is equivalent to $\sum_{-\infty}^{\infty} |h(k)| < \infty$. From the existence of the discrete-time Fourier transform, this stability implies that $H(e^{j\omega})$ exists, which further implies that the unit circle $|z| = 1$ must be in the ROC of $H(z)$. This result is called the *z-domain stability theorem*; therefore the dashed paths in Figure 4.9 exist only if the system is stable.

■ **THEOREM 2** ***z*-Domain LTI Stability**

An LTI system is stable if and only if the unit circle is in the ROC of $H(z)$.

For LTI causality we require that $h(n) = 0$, for $n < 0$ (i.e., a right-sided sequence). This implies that the ROC of $H(z)$ must be outside some circle of radius R_{h-}. This is not a sufficient condition since any right-sided sequence has a similar ROC. However, when the system is stable, then its causality is easy to check.

■ **THEOREM 3** ***z*-Domain Causal LTI Stability**

A causal LTI system is stable if and only if the system function $H(z)$ has all its poles inside the unit circle.

□ **EXAMPLE 4.13** A causal LTI system is described by the following difference equation:

$$y(n) = 0.81y(n-2) + x(n) - x(n-2)$$

Determine

a. the system function $H(z)$,
b. the unit impulse response $h(n)$,
c. the unit step response $v(n)$, that is, the response to the unit step $u(n)$, and
d. the frequency response function $H(e^{j\omega})$, and plot its magnitude and phase over $0 \le \omega \le \pi$.

Solution Since the system is causal, the ROC will be outside a circle with radius equal to the largest pole magnitude.

a. Taking the z-transform of both sides of the difference equation and then solving for $Y(z)/X(z)$ or using (4.20), we obtain

$$H(z) = \frac{1 - z^{-2}}{1 - 0.81z^{-2}} = \frac{1 - z^{-2}}{(1 + 0.9z^{-1})(1 - 0.9z^{-1})}, \quad |z| > 0.9$$

b. Using the MATLAB script for the partial fraction expansion,

```
>> b = [1,0,-1]; a = [1,0,-0.81]; [R,p,C] = residuez(b,a);
R =
  -0.1173
  -0.1173
p =
  -0.9000
   0.9000
C =
   1.2346
```

we have

$$H(z) = 1.2346 - 0.1173\frac{1}{1+0.9z^{-1}} - 0.1173\frac{1}{1-0.9z^{-1}}, \ |z| > 0.9$$

or from Table 4.1

$$h(n) = 1.2346\delta(n) - 0.1173\left\{1+(-1)^n\right\}(0.9)^n u(n)$$

c. From Table 4.1 $\mathcal{Z}[u(n)] = U(z) = \dfrac{1}{1-z^{-1}}$, $|z| > 1$. Hence

$$\begin{aligned} V(z) &= H(z)U(z) \\ &= \left[\frac{(1+z^{-1})(1-z^{-1})}{(1+0.9z^{-1})\,(1-0.9z^{-1})}\right]\left[\frac{1}{1-z^{-1}}\right], \quad |z| > 0.9 \cap |z| > 1 \\ &= \frac{1+z^{-1}}{(1+0.9z^{-1})\,(1-0.9z^{-1})}, \qquad |z| > 0.9 \end{aligned}$$

or

$$V(z) = 1.0556\frac{1}{1-0.9z^{-1}} - 0.0556\frac{1}{1+0.9z^{-1}}, \quad |z| > 0.9$$

Finally,

$$v(n) = \left[1.0556(0.9)^n - 0.0556\,(-0.9)^n\right]u(n)$$

Note that in the calculation of $V(z)$ there is a pole-zero cancellation at $z = 1$. This has two implications. First, the ROC of $V(z)$ is still $\{|z| > 0.9\}$ and not $\{|z| > 0.9 \cap |z| > 1 = |z| > 1\}$. Second, the step response $v(n)$ contains no steady-state term $u(n)$.

d. Substituting $z = e^{j\omega}$ in $H(z)$,

$$H(e^{j\omega}) = \frac{1 - e^{-j2\omega}}{1 - 0.81e^{-j2\omega}}$$

We will use the MATLAB script to compute and plot responses.

```
>> w = [0:1:500]*pi/500; H = freqz(b,a,w);
>> magH = abs(H); phaH = angle(H);
>> subplot(2,1,1); plot(w/pi,magH); grid
>> xlabel('frequency in pi units');  ylabel('Magnitude')
>> title('Magnitude Response')
>> subplot(2,1,2); plot(w/pi,phaH/pi); grid
>> xlabel('frequency in pi units'); ylabel('Phase in pi units')
>> title('Phase Response')
```

The frequency response plots are shown in Figure 4.10. □

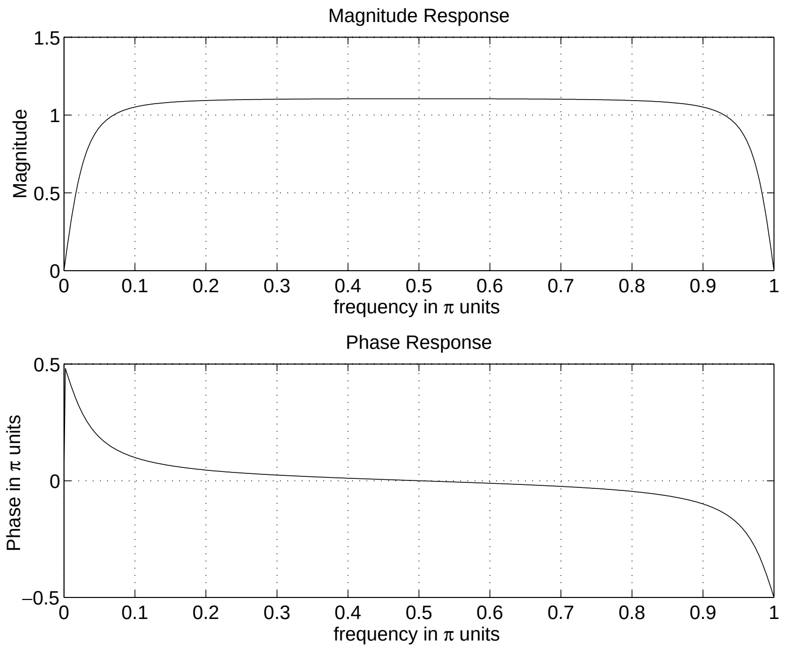


FIGURE 4.10 *Frequency response plots for Example 4.13*

4.5 SOLUTIONS OF THE DIFFERENCE EQUATIONS

In Chapter 2 we mentioned two forms for the solution of linear constant coefficient difference equations. One form involved finding the particular and the homogeneous solutions, while the other form involved finding the zero-input (initial condition) and the zero-state responses. Using z-transforms, we now provide a method for obtaining these forms. In addition, we will also discuss the *transient* and the *steady-state* responses. In digital signal processing, difference equations generally evolve in the positive n direction. Therefore our time frame for these solutions will be $n \geq 0$. For this purpose we define a version of the bilateral z-transform called the *one-sided z-transform.*

■ **DEFINITION 4** ***The One-sided z Transform***

The one-sided z-transform of a sequence $x(n)$ is given by

$$\mathcal{Z}^{+}[x(n)] \triangleq \mathcal{Z}\left[x(n)u(n)\right] \triangleq X^{+}\left[z\right] = \sum_{n=0}^{\infty} x(n)z^{-n} \tag{4.25}$$

Then the sample shifting property is given by

$$\begin{aligned}\mathcal{Z}^{+}[x(n-k)] &= \mathcal{Z}[x(n-k)u(n)] \\ &= \sum_{n=0}^{\infty} x(n-k)z^{-n} = \sum_{m=-k}^{\infty} x(m)z^{-(m+k)} \\ &= \sum_{m=-k}^{-1} x(m)z^{-(m+k)} + \left[\sum_{m=0}^{\infty} x(m)z^{-m}\right] z^{-k}\end{aligned}$$

or

$$\mathcal{Z}^{+}[x(n-k)] = x(-1)z^{1-k} + x(-2)z^{2-k} + \cdots + x(-k) + z^{-k}X^{+}(z) \quad \textbf{(4.26)}$$

This result can now be used to solve difference equations with nonzero initial conditions or with changing inputs. We want to solve the difference equation

$$1 + \sum_{k=1}^{N} a_k y(n-k) = \sum_{m=0}^{M} b_m x(n-m),\ n \geq 0$$

subject to these initial conditions:

$$\{y(i), i = -1, \ldots, -N\} \quad \text{and} \quad \{x(i), i = -1, \ldots, -M\}.$$

We now demonstrate its solution using an example.

□ **EXAMPLE 4.14** Solve

$$y(n) - \frac{3}{2}y(n-1) + \frac{1}{2}y(n-2) = x(n), \quad n \geq 0$$

where

$$x(n) = \left(\frac{1}{4}\right)^{n} u(n)$$

subject to $y(-1) = 4$ and $y(-2) = 10$.

Solution Taking the one-sided z-transform of both sides of the difference equation, we obtain

$$Y^{+}(z) - \frac{3}{2}[y(-1) + z^{-1}Y^{+}(z)] + \frac{1}{2}[y(-2) + z^{-1}y(-1) + z^{-2}Y^{+}(z)] = \frac{1}{1 - \frac{1}{4}z^{-1}}$$

Substituting the initial conditions and rearranging,

$$Y^{+}(z)\left[1 - \frac{3}{2}z^{-1} + \frac{1}{2}z^{-2}\right] = \frac{1}{1 - \frac{1}{4}z^{-1}} + (1 - 2z^{-1})$$

or

$$Y^+(z) = \frac{\dfrac{1}{1-\frac{1}{4}z^{-1}}}{1-\frac{3}{2}z^{-1}+\frac{1}{2}z^{-2}} + \frac{1-2z^{-1}}{1-\frac{3}{2}z^{-1}+\frac{1}{2}z^{-2}} \tag{4.27}$$

Finally,

$$Y^+(z) = \frac{2-\frac{9}{4}z^{-1}+\frac{1}{2}z^{-2}}{(1-\frac{1}{2}z^{-1})(1-z^{-1})(1-\frac{1}{4}z^{-1})}$$

Using the partial fraction expansion, we obtain

$$Y^+(z) = \frac{1}{1-\frac{1}{2}z^{-1}} + \frac{\frac{2}{3}}{1-z^{-1}} + \frac{\frac{1}{3}}{1-\frac{1}{4}z^{-1}} \tag{4.28}$$

After inverse transformation the solution is

$$y(n) = \left[\left(\frac{1}{2}\right)^n + \frac{2}{3} + \frac{1}{3}\left(\frac{1}{4}\right)^n\right]u(n) \tag{4.29}$$

□

Forms of the solutions The preceding solution is the *complete response* of the difference equation. It can be expressed in several forms.

- Homogeneous and particular parts:

$$y(n) = \underbrace{\left[\left(\frac{1}{2}\right)^n + \frac{2}{3}\right]u(n)}_{\text{Homogeneous part}} + \underbrace{\frac{1}{3}\left(\frac{1}{4}\right)^n u(n)}_{\text{Particular part}}$$

  The homogeneous part is due to the *system poles*, and the particular part is due to the *input poles.*

- Transient and steady-state responses:

$$y(n) = \underbrace{\left[\frac{1}{3}\left(\frac{1}{4}\right)^n + \left(\frac{1}{2}\right)^n\right]u(n)}_{\text{Transient response}} + \underbrace{\frac{2}{3}u(n)}_{\text{Steady-state response}}$$

  The transient response is due to poles that are *inside* the unit circle, whereas the steady-state response is due to poles that are *on* the unit circle. Note that when the poles are *outside* the unit circle, the response is termed an *unbounded* response.

- Zero-input (or initial condition) and zero-state responses:
  In equation (4.27) $Y^+(z)$ has two parts. The first part can be interpreted as

$$Y_{ZS}(z) = H(z)X(z)$$

while the second part as

$$Y_{ZI}(z) = H(z)X_{IC}(z)$$

where $X_{IC}(z)$ can be thought of as an equivalent *initial-condition input* that generates the same output Y_{ZI} as generated by the initial conditions. In this example $x_{IC}(n)$ is

$$x_{IC}(n) = \{\underset{\uparrow}{1}, -2\}$$

Now taking the inverse z-transform of each part of (4.27), we write the complete response as

$$y(n) = \underbrace{\left[\frac{1}{3}\left(\frac{1}{4}\right)^n - 2\left(\frac{1}{2}\right)^n + \frac{8}{3}\right]u(n)}_{\text{Zero-state response}} + \underbrace{\left[3\left(\frac{1}{2}\right)^n - 2\right]u(n)}_{\text{Zero-input response}}$$

From this example, it is clear that each part of the complete solution is, in general, a different function and emphasizes a different aspect of system analysis.

4.5.1 MATLAB IMPLEMENTATION

In Chapter 2 we used the `filter` function to solve the difference equation, given its coefficients and an input. This function can also be used to find the complete response when initial conditions are given. In this form the `filter` function is invoked by

```
y = filter(b,a,x,xic)
```

where `xic` is an equivalent initial-condition input array. To find the complete response in Example 4.14, we will use the MATLAB script

```
>>  n = [0:7]; x = (1/4).^n; xic = [1, -2];
>> format long;  y1 = filter(b,a,x,xic)
y1 =
  Columns 1 through 4
   2.00000000000000   1.25000000000000   0.93750000000000   0.79687500000000
  Columns 5 through 8
   0.73046875000000   0.69824218750000   0.68237304687500   0.67449951171875
>> y2 = (1/3)*(1/4).^n+(1/2).^n+(2/3)*ones(1,8) % MATLAB Check
y2 =
  Columns 1 through 4
   2.00000000000000   1.25000000000000   0.93750000000000   0.79687500000000
  Columns 5 through 8
   0.73046875000000   0.69824218750000   0.68237304687500   0.67449951171875
```

which agrees with the response given in (4.29). In Example 4.14 we computed $x_{IC}(n)$ analytically. However, in practice, and especially for large-order difference equations, it is tedious to determine $x_{IC}(n)$ analytically. MATLAB provides a function called `filtic`, which is available only in the Signal Processing toolbox. It is invoked by

```
xic = filtic(b,a,Y,X)
```

in which `b` and `a` are the filter coefficient arrays and `Y` and `X` are the initial-condition arrays from the initial conditions on $y(n)$ and $x(n)$, respectively, in the form

$$\begin{aligned}\texttt{Y} &= [y(-1),\ y(-2),\dots,\ y(-N)]\\ \texttt{X} &= [x(-1),\ x(-2),\dots,\ x(-M)]\end{aligned}$$

If $x(n) = 0, \quad n \le -1$ then `X` need not be specified in the `filtic` function. In Example 4.14 we could have used

```
>> Y = [4, 10];  xic = filtic(b,a,Y)
xic =
     1     -2
```

to determine $x_{IC}(n)$.

□ **EXAMPLE 4.15** Solve the difference equation

$$y(n) = \frac{1}{3}\left[x(n) + x(n-1) + x(n-2)\right] + 0.95y(n-1) - 0.9025y(n-2), \quad n \ge 0$$

where $x(n) = \cos(\pi n/3)u(n)$ and

$$y(-1) = -2,\ y(-2) = -3; \quad x(-1) = 1,\ x(-2) = 1$$

First determine the solution analytically and then by using MATLAB.

Solution Taking a one-sided z-transform of the difference equation

$$\begin{aligned}Y^+(z) = \frac{1}{3}&[X^+(z) + x(-1) + z^{-1}X^+(z) + x(-2) + z^{-1}x(-1) + z^{-2}X^+(z)]\\ &+ 0.95[y(-1) + z^{-1}Y^+(z)] - 0.9025[y(-2) + z^{-1}y(-1) + z^{-2}Y^+(z)]\end{aligned}$$

and substituting the initial conditions, we obtain

$$Y^+(z) = \frac{\frac{1}{3} + \frac{1}{3}z^{-1} + \frac{1}{3}z^{-2}}{1 - 0.95z^{-1} + 0.9025z^{-2}}X^+(z) + \frac{1.4742 + 2.1383z^{-1}}{1 - 0.95z^{-1} + 0.9025z^{-2}}$$

Clearly, $x_{IC}(n) = [1.4742, 2.1383]$. Now substituting $X^+(z) = \dfrac{1 - 0.5z^{-1}}{1 - z^{-1} + z^{-2}}$ and simplifying, we will obtain $Y^+(z)$ as a rational function. This simplification and further partial fraction expansion can be done using MATLAB.

MATLAB script:

```
>> b = [1,1,1]/3; a = [1,-0.95,0.9025];
>> Y = [-2,-3]; X = [1,1];  xic=filtic(b,a,Y,X)
xic =
    1.4742    2.1383
>> bxplus = [1,-0.5]; axplus = [1,-1,1]; % X(z) transform coeff.
>> ayplus = conv(a,axplus) % Denominator of Yplus(z)
ayplus =
    1.0000   -1.9500    2.8525   -1.8525    0.9025
>> byplus = conv(b,bxplus)+conv(xic,axplus)  % Numerator of Yplus(z)
byplus =
    1.8075    0.8308   -0.4975    1.9717
>> [R,p,C] = residuez(byplus,ayplus)
R =
   0.0584 + 3.9468i   0.0584 - 3.9468i   0.8453 + 2.0311i   0.8453 - 2.0311i
p =
   0.5000 - 0.8660i   0.5000 + 0.8660i   0.4750 + 0.8227i   0.4750 - 0.8227i
C =
     []
>> Mp = abs(p), Ap = angle(p)/pi % Polar form
Mp =
    1.0000    1.0000    0.9500    0.9500
Ap =
   -0.3333    0.3333    0.3333   -0.3333
```

Hence

$$\begin{aligned}
Y^+(z) &= \frac{1.8075 + 0.8308z^{-1} - 0.4975z^{-2} + 1.9717z^{-3}}{1 - 1.95z^{-1} + 2.8525z^{-2} - 1.8525z^{-3} + 0.9025z^{-4}} \\
&= \frac{0.0584 + j3.9468}{1 - e^{-j\pi/3}z^{-1}} + \frac{0.0584 - j3.9468}{1 - e^{j\pi/3}z^{-1}} \\
&\quad + \frac{0.8453 + j2.0311}{1 - 0.95e^{j\pi/3}z^{-1}} + \frac{0.8453 - j2.0311}{1 - 0.95e^{-j\pi/3}z^{-1}}
\end{aligned}$$

Now from Table 4.1

$$\begin{aligned}
y(n) &= (0.0584 + j3.9468)\, e^{-j\pi n/3} + (0.0584 - j3.9468)\, e^{j\pi n/3} \\
&\quad + (0.8453 + j2.031)\,(0.95)^n\, e^{j\pi n/3} + (0.8453 - j2.031)\,(0.95)^n\, e^{-j\pi n/3} \\
&= 0.1169\cos(\pi n/3) + 7.8937\sin(\pi n/3) \\
&\quad + (0.95)^n\,[1.6906\cos(\pi n/3) - 4.0623\sin(\pi n/3)]\,, \quad n \geq 0
\end{aligned}$$

The first two terms of $y(n)$ correspond to the steady-state response, as well as to the particular response, while the last two terms are the transient response (and homogeneous response) terms.

To solve this example using MATLAB, we will need the `filtic` function, which we have already used to determine the $x_{IC}(n)$ sequence. The solution will be a numerical one. Let us determine the first 8 samples of $y(n)$.

MATLAB script:

```
>> n = [0:7]; x = cos(pi*n/3);  y = filter(b,a,x,xic)
y =
  Columns 1 through 4
   1.80750000000000   4.35545833333333   2.83975000000000  -1.56637197916667
  Columns 5 through 8
  -4.71759442187500  -3.40139732291667   1.35963484230469   5.02808085078841
% Matlab Verification
>> A=real(2*R(1)); B=imag(2*R(1)); C=real(2*R(3)); D=imag(2*R(4));
>> y=A*cos(pi*n/3)+B*sin(pi*n/3)+((0.95).^n).*(C*cos(pi*n/3)+D*sin(pi*n/3))
y =
  Columns 1 through 4
   1.80750000000048   4.35545833333359   2.83974999999978  -1.56637197916714
  Columns 5 through 8
  -4.71759442187528  -3.40139732291648   1.35963484230515   5.02808085078871
```
□

4.6 PROBLEMS

P4.1 Determine the z-transform of the following sequences using the definition (4.1). Indicate the region of convergence for each sequence and verify the z-transform expression using MATLAB.

1. $x(n) = \{3, 2, \underset{\uparrow}{1}, -2, -3\}$.
2. $x(n) = (0.8)^n u(n-2)$. Verify the z-transform expression using MATLAB.
3. $x(n) = [(0.5)^n + (-0.8)^n]u(n)$. Verify the z-transform expression using MATLAB.
4. $x(n) = 2^n \cos(0.4\pi n)u(-n)$.
5. $x(n) = (n+1)(3)^n u(n)$. Verify the z-transform expression using MATLAB.

P4.2 Consider the sequence $x(n) = (0.9)^n \cos(\pi n/4)u(n)$. Let

$$y(n) = \begin{cases} x(n/2), & n = 0, \pm 2, \pm 4, \cdots; \\ 0, & \text{otherwise}. \end{cases}$$

1. Show that the z-transform $Y(z)$ of $y(n)$ can be expressed in terms of the z-transform $X(z)$ of $x(n)$ as $Y(z) = X(z^2)$.
2. Determine $Y(z)$.
3. Using MATLAB, verify that the sequence $y(n)$ has the z-transform $Y(z)$.

P4.3 Determine the z-transform of the following sequences using the z-transform table and the z-transform properties. Express $X(z)$ as a rational function in z^{-1}. Verify your results using MATLAB. Indicate the region of convergence in each case, and provide a pole-zero plot.

1. $x(n) = 2\delta(n-2) + 3u(n-3)$
2. $x(n) = 3(0.75)^n \cos(0.3\pi n)u(n) + 4(0.75)^n \sin(0.3\pi n)u(n)$
3. $x(n) = n\sin(\frac{\pi n}{3})u(n) + (0.9)^n u(n-2)$

4. $x(n) = n^2(2/3)^{n-2}u(n-1)$
5. $x(n) = (n-3)(\frac{1}{4})^{n-2}\cos\{\frac{\pi}{2}(n-1)\}u(n)$

P4.4 Let $x(n)$ be a complex-valued sequence with the real part $x_{\text{R}}(n)$ and the imaginary part $x_{\text{I}}(n)$.

1. Prove the following z-transform relations:

$$X_{\text{R}}(z) \triangleq \mathcal{Z}[x_{\text{R}}(n)] = \frac{X(z) + X^*(z^*)}{2} \quad \text{and} \quad X_{\text{I}}(z) \triangleq \mathcal{Z}[x_{\text{I}}(n)] = \frac{X(z) - X^*(z^*)}{2}$$

2. Verify these relations for $x(n) = \exp\{(-1 + j0.2\pi)n\}\,u(n)$.

P4.5 The z-transform of $x(n)$ is $X(z) = 1/(1 + 0.5z^{-1})$, $|z| \geq 0.5$. Determine the z-transforms of the following sequences and indicate their region of convergence.

1. $x_1(n) = x(3-n) + x(n-3)$
2. $x_2(n) = (1 + n + n^2)x(n)$
3. $x_3(n) = (\frac{1}{2})^n x(n-2)$
4. $x_4(n) = x(n+2) * x(n-2)$
5. $x_5(n) = \cos(\pi n/2)x^*(n)$

P4.6 Repeat Problem P4.5 if

$$X(z) = \frac{1 + z^{-1}}{1 + \frac{5}{6}z^{-1} + \frac{1}{6}z^{-2}};\ |z| > \frac{1}{2}$$

P4.7 The inverse z-transform of $X(z)$ is $x(n) = (1/2)^n u(n)$. Using the z-transform properties, determine the sequences in each of the following cases.

1. $X_1(z) = \frac{z-1}{z}X(z)$
2. $X_2(z) = zX(z^{-1})$
3. $X_3(z) = 2X(3z) + 3X(z/3)$
4. $X_4(z) = X(z)X(z^{-1})$
5. $X_5(z) = z^2\frac{dX(z)}{dz}$

P4.8 If sequences $x_1(n)$, $x_2(n)$, and $x_3(n)$ are related by $x_3(n) = x_1(n) * x_2(n)$, then

$$\sum_{n=-\infty}^{\infty} x_3(n) = \left(\sum_{n=-\infty}^{\infty} x_1(n)\right)\left(\sum_{n=-\infty}^{\infty} x_2(n)\right)$$

1. Prove this result by substituting the definition of convolution in the left-hand side.
2. Prove this result using the convolution property.
3. Verify this result using MATLABand choosing any two random sequences $x_1(n)$, and $x_2(n)$.

P4.9 Determine the results of the following polynomial operations using MATLAB.

1. $X_1(z) = (1 - 2z^{-1} + 3z^{-2} - 4z^{-3})(4 + 3z^{-1} - 2z^{-2} + z^{-3})$
2. $X_2(z) = (z^2 - 2z + 3 + 2z^{-1} + z^{-2})(z^3 - z^{-3})$
3. $X_3(z) = (1 + z^{-1} + z^{-2})^3$
4. $X_4(z) = X_1(z)X_2(z) + X_3(z)$
5. $X_5(z) = (z^{-1} - 3z^{-3} + 2z^{-5} + 5z^{-7} - z^{-9})(z + 3z^2 + 2z^3 + 4z^4)$

P4.10 The `deconv` function is useful in dividing two causal sequences. Write a MATLAB function `deconv_m` to divide two noncausal sequences (similar to the `conv` function). The format of this function should be

```
function [p,np,r,nr] = deconv_m(b,nb,a,na)
% Modified deconvolution routine for noncausal sequences
% function [p,np,r,nr] = deconv_m(b,nb,a,na)
%
%  p = polynomial part of support np1 <= n <= np2
% np = [np1, np2]
%  r = remainder part of support nr1 <= n <= nr2
% nr = [nr1, nr2]
%  b = numerator polynomial of support nb1 <= n <= nb2
% nb = [nb1, nb2]
%  a = denominator polynomial of support na1 <= n <= na2
% na = [na1, na2]
%
```

Check your function on the following operartion

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

P4.11 Determine the following inverse z-transforms using the partial fraction expansion method.

1. $X_1(z) = (1-z^{-1}-4z^{-2}+4z^{-3})/(1-\frac{11}{4}z^{-1}+\frac{13}{8}z^{-2}-\frac{1}{4}z^{-3})$. The sequence is rightsided.
2. $X_2(z) = (1+z^{-1}-4z^{-2}+4z^{-3})/(1-\frac{11}{4}z^{-1}+\frac{13}{8}z^{-2}-\frac{1}{4}z^{-3})$. The sequence is absolutely summable.
3. $X_3(z) = (z^3-3z^2+4z+1)/(z^3-4z^2+z-0.16)$. The sequence is leftsided.
4. $X_4(z) = z/(z^3+2z^2+1.25z+0.25)$, $|z|>1$
5. $X_5(z) = z/(z^2-0.25)^2$, $|z|<0.5$

P4.12 Consider the sequence

$$x(n) = A_c(r)^n\cos(\pi v_0 n)u(n) + A_s(r)^n\sin(\pi v_0 n)u(n) \tag{4.30}$$

The z-transform of this sequence is a 2-order (proper) rational function that contains a complex-conjugate pole pair. The objective of this problem is to develop a MATLAB function that can be used to obtain the inverse z-transform of such a rational function so that the inverse does not contain any complex numbers.

1. Show that the z-transform of $x(n)$ in (4.30) is given by

$$X(z) = \frac{b_0+b_1z^{-1}}{1+a_1z^{-1}+a_2z^{-2}}; \quad |z|>|r| \tag{4.31}$$

where

$$b_0 = A_c;\ b_1 = r[A_s\sin(\pi v_0) - A_c\cos(\pi v_0)];\ a_1 = -2r\cos(\pi v_0);\ a_2 = r^2 \tag{4.32}$$

2. Using (4.32), determine the signal parameters A_c, A_s, r, and v_0 in terms of the rational function parameters b_0, b_1, a_1, and a_2.
3. Using your results in part b above, design a MATLABfunction, `invCCPP`, that computes signal parameters using the rational function parameters. The format of this function should be:

```
function [As,Ac,r,v0] = invCCPP(b0,b1,a1,a2)
```

P4.13 Suppose $X(z)$ is given as follows:

$$X(z) = \frac{2+3z^{-1}}{1-z^{-1}+0.81z^{-2}}, \; |z| > 0.9$$

1. Using the MATLABfunction `invCCPP` given in Problem P4.12, determine $x(n)$ in a form that contains no complex numbers.
2. Using MATLAB, compute the first 20 samples of $x(n)$, and compare them with your answer in the above part.

P4.14 The z-transform of a causal sequence is given as

$$X(z) = \frac{-2+5.65z^{-1}-2.88z^{-2}}{1-0.1z^{-1}+0.09z^{-2}+0.648z^{-3}} \tag{4.33}$$

which contains a complex-conjugate pole pair as well as a real-valued pole.

1. Using the `residuez` function express (4.33) as

$$X(z) = \frac{(\quad)+(\quad)z^{-1}}{1+(\quad)z^{-1}+(\quad)z^{-2}} + \frac{(\quad)}{1+(\quad)z^{-1}} \tag{4.34}$$

   Note that you will have to use the `residuez` function in both directions.
2. Now using your function `invCCPP` and the inverse of the real-valued pole factor, determine the causal sequence $x(n)$ from the $X(z)$ in (4.34) so that it contains no complex numbers.

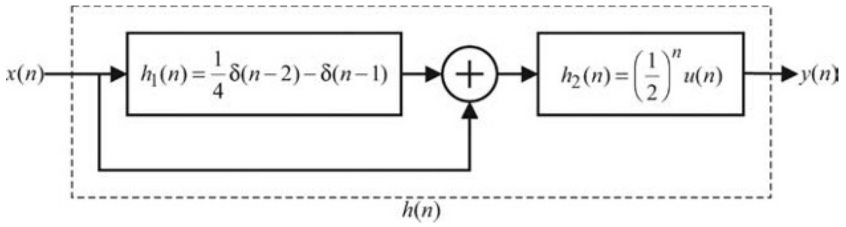
P4.15 For the linear and time-invariant systems described by the following impulse responses, determine (i) the system function representation, (ii) the difference equation representation, (iii) the pole-zero plot, and (iv) the output $y(n)$ if the input is $x(n) = \left(\frac{1}{4}\right)^n u(n)$.

1. $h(n) = 5(1/4)^n u(n)$
2. $h(n) = n(1/3)^n u(n) + (-1/4)^n u(n)$
3. $h(n) = 3(0.9)^n \cos(\pi n/4 + \pi/3)u(n+1)$
4. $h(n) = \dfrac{(0.5)^n \sin[(n+1)\pi/3]}{\sin(\pi/3)} u(n)$
5. $h(n) = [2 - \sin(\pi n)]u(n)$

P4.16 Consider the system shown below.

1. Using the z-transform approach, show that the impulse response, $h(n)$, of the overall system is given by

$$h(n) = \delta(n) - \frac{1}{2}\delta(n-1)$$


$x(n)$
$h_1(n) = \frac{1}{4}\delta(n-2) - \delta(n-1)$
$h_2(n) = \left(\frac{1}{2}\right)^n u(n)$
$y(n)$
$h(n)$

2. Determine the difference equation representation of the overall system that relates the output $y(n)$ to the input $x(n)$.
3. Is this system causal? BIBO stable? Explain clearly to receive full credit.
4. Determine the frequency response $H(e^{j\omega})$ of the overall system.
5. Using MATLAB, provide a plot of this frequency response over $0 \le \omega \le \pi$.

P4.17 For the linear and time-invariant systems described by the following system functions, determine (i) the impulse response representation, (ii) the difference equation representation, (iii) the pole-zero plot, and (iv) the output $y(n)$ if the input is $x(n) = 3\cos(\pi n/3)u(n)$.

1. $H(z) = (z+1)/(z-0.5)$, causal system
2. $H(z) = (1+z^{-1}+z^{-2})/(1+0.5z^{-1}-0.25z^{-2})$, stable system
3. $H(z) = (z^2-1)/(z-3)^2$, anticausal system
4. $H(z) = \dfrac{z}{z-0.25} + \dfrac{1-0.5z^{-1}}{1+2z^{-1}}$, stable system
5. $H(z) = (1+z^{-1}+z^{-2})^2$

P4.18 For the linear, causal, and time-invariant systems described by the following difference equations, determine (i) the impulse response representation, (ii) the system function representation, (iii) the pole-zero plot, and (iv) the output $y(n)$ if the input is $x(n) = 2(0.9)^n u(n)$.

1. $y(n) = [x(n) + 2x(n-1) + x(n-3)]/4$
2. $y(n) = x(n) + 0.5x(n-1) - 0.5y(n-1) + 0.25y(n-2)$
3. $y(n) = 2x(n) + 0.9y(n-1)$
4. $y(n) = -0.45x(n) - 0.4x(n-1) + x(n-2) + 0.4y(n-1) + 0.45y(n-2)$
5. $y(n) = \sum_{m=0}^{4}(0.8)^m x(n-m) - \sum_{\ell=1}^{4}(0.9)^\ell y(n-\ell)$

P4.19 The output sequence $y(n)$ in Problem P4.18 is the total response. For each of the systems given in Problem P4.18, separate $y(n)$ into (i) the homogeneous part, (ii) the particular part, (iii) the transient response, and (iv) the steady-state response.

P4.20 A stable system has four zeros and four poles as given here:

$$\text{zeros: } \pm 1,\ \pm j1 \qquad \text{Poles: } \pm 0.9,\ \pm j0.9$$

It is also known that the frequency response function $H(e^{j\omega})$ evaluated at $\omega = \pi/4$ is equal to 1, i.e.,

$$H(e^{j\pi/4}) = 1$$

1. Determine the system function $H(z)$, and indicate its region of convergence.
2. Determine the difference equation representation.
3. Determine the steady-state response $y_{ss}(n)$ if the input is $x(n) = \cos(\pi n/4)u(n)$.
4. Determine the transient response $y_{tr}(n)$ if the input is $x(n) = \cos(\pi n/4)u(n)$.

P4.21 A digital filter is described by the frequency response function

$$H(e^{j\omega}) = [1 + 2\cos(\omega) + 3\cos(2\omega)]\cos\left(\frac{\omega}{2}\right)e^{-j5\omega/2}$$

1. Determine the difference equation representation.
2. Using the `freqz` function, plot the magnitude and phase of the frequency response of the filter. Note the magnitude and phase at $\omega = \pi/2$ and at $\omega = \pi$.
3. Generate 200 samples of the signal $x(n) = \sin(\pi n/2) + 5\cos(\pi n)$, and process through the filter to obtain $y(n)$. Compare the steady-state portion of $y(n)$ to $x(n)$. How are the amplitudes and phases of two sinusoids affected by the filter?

P4.22 Repeat Problem 4.21 for the following filter

$$H(e^{j\omega}) = \frac{1 + e^{-j4\omega}}{1 - 0.8145e^{-j4\omega}}$$

P4.23 Solve the following difference equation for $y(n)$ using the one-sided z-transform approach.

$$\begin{aligned} y(n) &= 0.81y(n-2) + x(n) - x(n-1),\ n \geq 0; \quad y(-1) = 2,\ y(-2) = 2 \\ x(n) &= (0.7)^n u(n+1) \end{aligned}$$

Generate the first 20 samples of $y(n)$ using MATLAB, and compare them with your answer.

P4.24 Solve the difference equation for $y(n)$, $n \geq 0$

$$y(n) - 0.4y(n-1) - 0.45y(n-2) = 0.45x(n) + 0.4x(n-1) - x(n-2)$$

driven by the input $x(n) = \left[2 + \left(\frac{1}{2}\right)^n\right]u(n)$ and subject to

$$y(-1) = 0,\ y(-2) = 3;\ x(-1) = x(-2) = 2$$

Decompose the solution $y(n)$ into (i) transient response, (ii) steady-state response, (iii) zero-input response, and (iv) zero-state response.

P4.25 A stable, linear and time-invariant system is given by the following system function

$$H(z) = \frac{4z^2 - 2\sqrt{2}z + 1}{z^2 - 2\sqrt{2}z + 4}$$

1. Determine the difference equation representation for this system.
2. Plot the poles and zeros of $H(z)$, and indicate the ROC.
3. Determine the unit sample response $h(n)$ of this system.
4. Is this system causal? If the answer is yes, justify it. If the answer is no, find a causal unit sample response that satisfies the system function.

P4.26 Determine the zero-input, zero-state, and steady-state responses of the system

$$y(n) = 0.9801y(n-2) + x(n) + 2x(n-1) + x(n-2),\ n \geq 0; \quad y(-2) = 1,\ y(-1) = 0$$

to the input $x(n) = 5(-1)^n u(n)$.