7

Two-Dimensional Signal Processing

Jae S. Lim
Massachusetts Institute of Technology

7.0 INTRODUCTION

At a conceptual level, there is a great deal of similarity between two-dimensional (2-D) signal processing and one-dimensional (1-D) signal processing. In 1-D signal processing, the concepts discussed are filtering, Fourier transforms, discrete Fourier transforms, fast Fourier transforms, etc. In 2-D signal processing, we again are concerned with concepts such as filtering, Fourier transforms, discrete Fourier transforms, and fast Fourier transforms. As a consequence, the general concepts that we develop in 2-D signal processing can be viewed, in many cases, as straightforward extensions of the results in 1-D signal processing.

At a more detailed level, however, considerable differences exist between 1-D and 2-D signal processing. One major difference is the amount of data involved in typical applications. In speech processing, an important 1-D signal processing application, speech is typically sampled at a 10-kHz rate and we have 10,000 data points to process in a second. However, in video processing, where processing an image frame is an important 2-D signal processing application, we may have 30 frames/s, with each frame consisting of 500×500 pixels (picture elements). In this case, we would have 7.5 million data points to process per second, which is orders of magnitude greater than the case of speech processing. Due to this difference in data rate requirements, the computational efficiency of a signal processing algorithm plays a much more important role in 2-D signal processing, and advances in hardware tech-

Jae S. Lim is with the Research Laboratory of Electronics and the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. This work has been supported in part by the National Science Foundation under Grant ECS-8407285 and in part by the Advanced Research Projects Agency monitored by ONR under contract N00014-81-K-0742.

nology will have a much greater impact on 2-D signal processing applications in the future.

Another major difference comes from the fact that there is less complete mathematics for 2-D signal processing than for 1-D signal processing. For example, many 1-D systems are described by differential equations, while many 2-D systems can be described by partial differential equations. We know a great deal more about differential equations than about partial differential equations. Another example is the absence of the fundamental theorem of algebra for 2-D polynomials. For 1-D polynomials, the fundamental theorem of algebra states that any 1-D polynomial can be factored as a product of first-order polynomials. A 2-D polynomial, however, generally cannot be factored as a product of lower-order polynomials. This difference has a major impact on many results in signal processing. For example, an important structure for realizing a 1-D digital filter is the cascade structure. In the cascade structure, the z-transform of the digital filter unit sample response is factored as a product of lower-order polynomials and the realizations of these lower-order factors are cascaded. The z-transform of a 2-D digital filter unit sample response cannot, in general, be factored as a product of lower-order polynomials and the cascade structure therefore is not a general structure for a 2-D digital filter realization. Another consequence of the nonfactorability of a 2-D polynomial is the difficulty associated with issues related to the system stability. In a 1-D system, the pole locations can be easily determined, and an unstable system can be stabilized without affecting the magnitude response by simple manipulation of pole locations. In a 2-D system, because poles are surfaces rather than points and because of the absence of the fundamental theorem of algebra, it is extremely difficult to determine the pole locations. As a result, checking the stability of a 2-D system and stabilizing an unstable 2-D system without affecting the magnitude response is extremely difficult.

Another difference between 1-D and 2-D signal processing is the notion of causality. In a typical 1-D application such as speech processing, a system is generally required to be causal to avoid delay since there is a well-defined notion of past, present, and future. In a typical 2-D application such as image processing, it may not be necessary to impose the causality constraint. An image frame, for example, can be processed from top to bottom, from left to right, in diagonal directions, etc. As a result, there is generally more flexibility in designing a 2-D system than in designing a 1-D system.

As we have seen, there is considerable similarity and at the same time considerable difference between 1-D and 2-D signal processing. In this chapter, we will study the results in 1-D signal processing that can be extended to 2-D signal processing. Our discussion will rely heavily on the reader's knowledge of 1-D signal processing theory (a brief summary of which is in the Appendix to this book). We will also study, with much greater emphasis, the results in 2-D signal processing that are significantly different from those in 1-D signal processing. We will study what the differences are, where they come from, and what impacts they have on 2-D signal processing applications. Since we will study the similarities and differences between 1-D and 2-D signal processing and since 1-D signal processing is a special case of 2-D signal processing, this chapter will help us understand not only 2-D signal processing theories but also 1-D signal processing theories at a much deeper level.

7.1 SIGNALS AND SYSTEMS

7.1.1 Signals

The signals that we deal with in developing 2-D signal processing theories are discrete space signals, which are discrete in space[†] and continuous in amplitude. Discrete space signals are also referred to as *sequences*.

A two-dimensional (2-D) discrete space signal (sequence) will be denoted by functions whose two arguments are integers. For example, $x(n_1, n_2)$ represents a sequence that is defined for all integer values of n_1 and n_2. Note that $x(n_1, n_2)$ for a noninteger n_1 or n_2 is not zero but is undefined. The notation $x(n_1, n_2)$ refers to the discrete space function x or to the value of the function x at a specific (n_1, n_2). The distinction between these two will be obvious from the context.

One reasonable way to sketch a sequence $x(n_1, n_2)$ is to use a three-dimensional (3-D) perspective plot, where the height at (n_1, n_2) represents the amplitude at (n_1, n_2). Sketching a 3-D perspective plot, however, is often very tedious. An alternative way to sketch a 2-D sequence, which we'll use in this chapter, is with a 2-D plot where open circles represent the amplitude of 0 and the solid circles represent nonzero amplitudes, with the value in parentheses representing the amplitude. An example of a 2-D sequence sketched in this way is shown in Fig. 7.1. In this figure, $x(3, 1)$ is 0 and $x(1, 1)$ is 2. Many sequences that we use will have amplitudes 0 or 1 for large regions of (n_1, n_2). In such instances, for convenience, the open circles and parentheses will be eliminated. If there is neither an open circle nor a filled circle at a particular (n_1, n_2), the sequence has zero amplitude at that point. If there is a filled circle with

[†]Even though we refer to "space," the independent variable can represent other quantities, such as time.

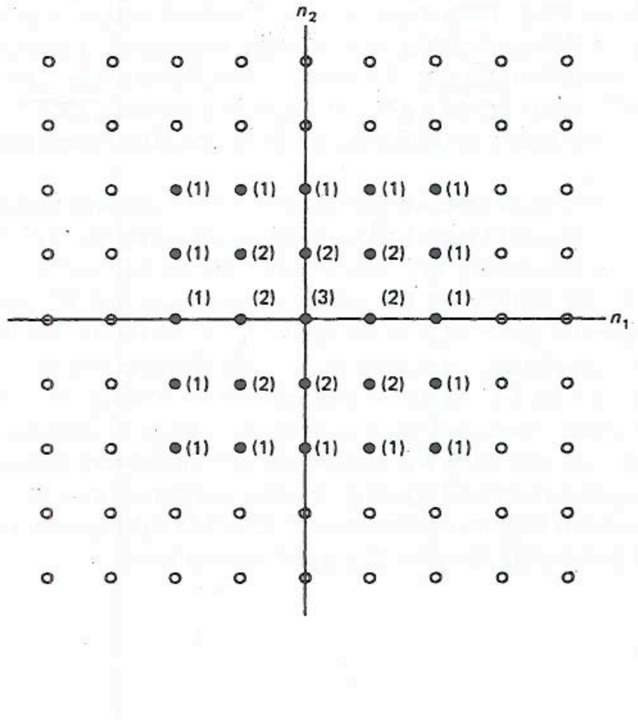


Figure 7.1 An example of a 2-D sequence. Open circles represent the amplitude of 0, and solid circles represent nonzero amplitudes, with the values in parentheses representing the amplitude.

no amplitude specification at a particular (n_1, n_2) then the sequence has amplitude 1 at that point. Fig. 7.2 shows the result when this additional simplification is made to the sequence in Fig. 7.1.

Some sequences and classes of sequences that play a particularly important role in 2-D digital signal processing are discussed next.

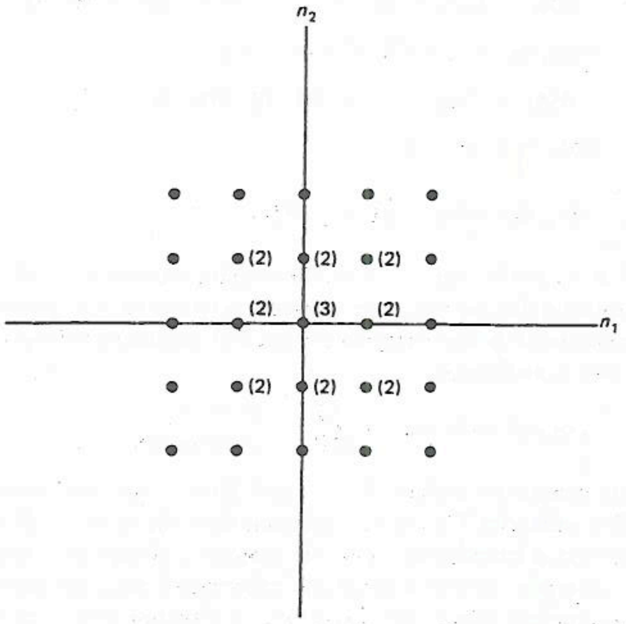


Figure 7.2 The sequence in Fig. 7.1 sketched with some simplifications. The open circles have been eliminated, and solid circles with amplitude of 1 have no amplitude specifications.

Impulses. The impulse or unit sample sequence, denoted by $\delta(n_1, n_2)$, is defined as

$$\delta(n_1, n_2) = \begin{cases} 1, & n_1 = n_2 = 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

The sequence $\delta(n_1, n_2)$, sketched in Fig. 7.3, plays a role similar to the impulse $\delta(n)$ in 1-D signal processing.

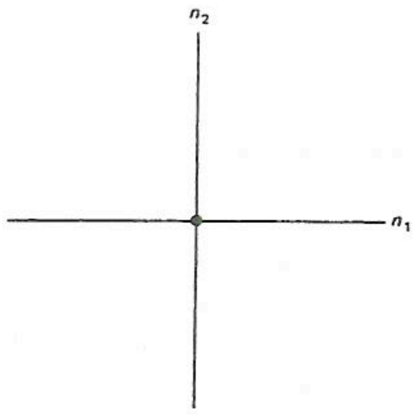


Figure 7.3 Unit sample sequence $\delta(n_1, n_2)$.

Any sequence $x(n_1, n_2)$ can be represented as a linear combination of delayed impulses as follows:

$$\begin{aligned} x(n_1, n_2) = & \cdots + x(-1,-1)\cdot\delta(n_1+1, n_2+1) + x(0,-1)\cdot\delta(n_1, n_2+1) \\ & + x(1,-1)\cdot\delta(n_1-1, n_2+1) + \cdots + x(-1,0)\cdot\delta(n_1+1, n_2) \\ & + x(0,0)\cdot\delta(n_1, n_2) + x(1,0)\cdot\delta(n_1-1, n_2) + \cdots \\ & + x(-1,1)\cdot\delta(n_1+1, n_2-1) + x(0,1)\cdot\delta(n_1, n_2-1) \\ & + x(1,1)\cdot\delta(n_1-1, n_2-1) + \cdots \\ = & \sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty} x(k_1, k_2)\cdot\delta(n_1-k_1, n_2-k_2) \end{aligned} \tag{7.2}$$

The representation of $x(n_1, n_2)$ by Eq. (7.2) is very useful in system analysis.

One class of impulses that do not have any counterparts in 1-D processing are line impulses. An example of a line impulse is the 2-D sequence $\delta_T(n_1)$, which is sketched in Fig. 7.4 and is defined as

$$x(n_1, n_2) = \delta_T(n_1) = \begin{cases} 1, & n_1 = 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.3}$$

Other examples of line impulses include $\delta_T(n_2)$ and $\delta(n_1 - n_2)$, which are defined similarly to $\delta_T(n_1)$. The subscript T in $\delta_T(n_1)$ indicates that $\delta_T(n_1)$ is a 2-D sequence. When the 2-D sequence is a function of only one variable, it may be confused with a 1-D sequence. For example, $\delta_T(n_1)$ without the subscript T may be confused with the 1-D unit sample sequence $\delta(n_1)$. To avoid this confusion, whenever a 2-D sequence is a function of one variable, the subscript T will be used to note that it is a 2-D sequence. The sequence $x_T(n_1)$, for example, is a 2-D sequence, while $x(n_1)$ is a 1-D sequence.

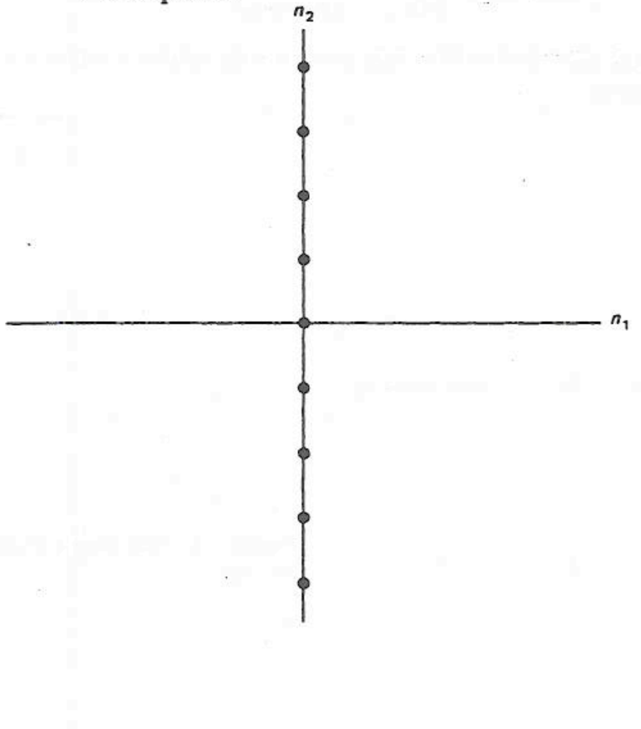


Figure 7.4 Line impulse $\delta_T(n_1)$.

Step Sequences. The unit step sequence, denoted by $u(n_1, n_2)$, is defined as

$$u(n_1, n_2) = \begin{cases} 1, & n_1, n_2 \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.4}$$

The sequence $u(n_1, n_2)$, sketched in Fig. 7.5, is related to $\delta(n_1, n_2)$ as

$$u(n_1, n_2) = \sum_{k_1=-\infty}^{n_1} \sum_{k_2=-\infty}^{n_2} \delta(k_1, k_2) \tag{7.5}$$

or

$$\delta(n_1, n_2) = u(n_1, n_2) - u(n_1 - 1, n_2) - u(n_1, n_2 - 1) + u(n_1 - 1, n_2 - 1) \tag{7.6}$$

Some step sequences do not have any counterparts in 1-D processing, such as the 2-D sequence $u_T(n_1)$, which is defined as

$$x(n_1, n_2) = u_T(n_1) = \begin{cases} 1, & n_1 \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{7.7}$$

Other examples include $u_T(n_2)$ and $u(n_1 - n_2)$, which are defined similarly to $u_T(n_1)$.

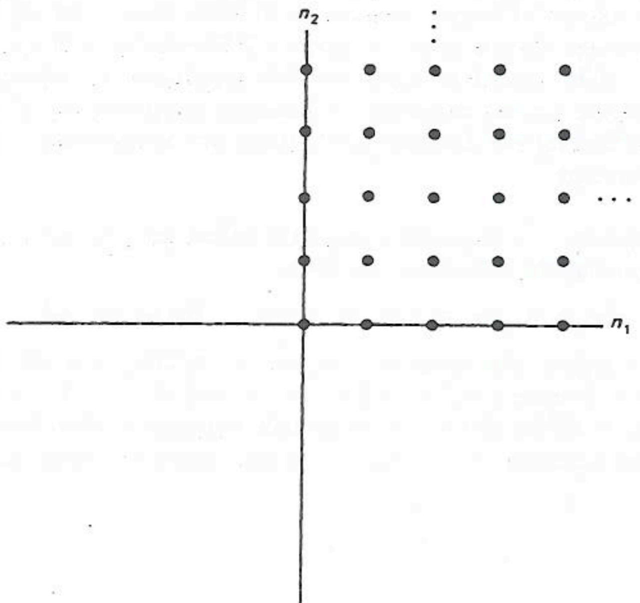


Figure 7.5 Unit step sequence $u(n_1, n_2)$.

Exponential Sequences. Exponential sequences of the type $A \cdot \alpha^{n_1} \cdot \beta^{n_2}$ are an important class of sequences for system analysis. As we will see later, these sequences are eigenfunctions of linear shift-invariant systems.

Separable Sequences. A 2-D sequence $x(n_1, n_2)$ is called a separable sequence if it can be expressed in the form

$$x(n_1, n_2) = f(n_1) \cdot g(n_2) \tag{7.8}$$

where $f(n_1)$ is a function of only n_1, and $g(n_2)$ is a function of only n_2. Even though it is possible to view $f(n_1)$ and $g(n_2)$ as 2-D sequences, it is more convenient to consider them as 1-D sequences. For that reason, we use the notations $f(n_1)$ and $g(n_2)$ rather than $f_T(n_1)$ and $g_T(n_2)$.

The unit sample sequence $\delta(n_1, n_2)$ is a separable sequence since $\delta(n_1, n_2)$ can be expressed as

$$\delta(n_1, n_2) = \delta(n_1) \cdot \delta(n_2) \tag{7.9}$$

where $\delta(n_1)$ and $\delta(n_2)$ are 1-D unit sample sequences. Other examples of separable sequences include $u(n_1, n_2)$ and $a^{n_1} \cdot b^{n_2} + b^{n_1+n_2}$, which can be written as $(a^{n_1} + b^{n_1}) \cdot b^{n_2}$.

Separable sequences form a very special class of 2-D sequences; a typical 2-D sequence is not, in general, a separable sequence. To illustrate this, we consider a sequence $x(n_1, n_2)$ that is zero outside $0 \le n_1 \le N_1 - 1$ and $0 \le n_2 \le N_2 - 1$. A general sequence $x(n_1, n_2)$ of this type has $N_1 \cdot N_2$ degrees of freedom. If $x(n_1, n_2)$ is a separable sequence, $x(n_1, n_2)$ is completely specified by some $f(n_1)$ that is zero outside $0 \le n_1 \le N_1 - 1$ and some $g(n_2)$ that is zero outside $0 \le n_2 \le N_2 - 1$; consequently, it has only $N_1 + N_2$ degrees of freedom.

Even though separable sequences form a very special class of 2-D sequences, they play an important role in 2-D signal processing. In those cases when the results that apply to 1-D sequences do not extend to general 2-D sequences in a straightforward manner, they often extend to separable 2-D sequences. In addition, the separability of a sequence can be exploited in reducing computations in various contexts such as digital filtering and discrete Fourier transform computation. This will be discussed in later sections.

Periodic Sequences. A sequence $x(n_1, n_2)$ is called periodic with a period $N_1 \times N_2$ if $x(n_1, n_2)$ satisfies the following condition:

$$x(n_1, n_2) = x(n_1 + N_1, n_2) = x(n_1, n_2 + N_2) \qquad \text{for all } (n_1, n_2) \tag{7.10}$$

where N_1 and N_2 are integers. For example, $\cos[\pi n_1 + (\pi/2)n_2]$ is a periodic sequence with period 2×4 since $\cos[\pi n_1 + (\pi/2)n_2] = \cos[\pi(n_1 + 2) + (\pi/2)n_2] = \cos[\pi n_1 + (\pi/2)(n_2 + 4)]$ for all (n_1, n_2). A periodic sequence is often denoted by adding a ~ (tilde) on the sequence, e.g., $\tilde{x}(n_1, n_2)$, to distinguish it from an aperiodic sequence.

7.1.2 Systems

If there is a unique output for any given input, the input-output mapping is called a system. A system T that maps an input $x(n_1, n_2)$ to an output $y(n_1, n_2)$ is represented by

$$y(n_1, n_2) = \mathrm{T}[x(n_1, n_2)] \tag{7.11}$$

This definition of a system is very broad. Without some restrictions, the characterization of a system requires a complete input-output relationship—knowing the output of a system to a certain set of inputs does not allow us to determine the output of the system to other sets of inputs. Two types of restrictions that greatly simplify the characterization and analysis of a system are linearity and shift invariance. Fortunately, many systems in practice can often be approximated by a linear and shift-invariant system.

The linearity of a system T is defined as

$$\text{Linearity} \quad \longleftrightarrow \quad \mathrm{T}[a \cdot x_1(n_1, n_2) + b \cdot x_2(n_1, n_2)] = a \cdot y_1(n_1, n_2) + b \cdot y_2(n_1, n_2) \tag{7.12}$$

where $\mathrm{T}[x_1(n_1, n_2)] = y_1(n_1, n_2)$, $\mathrm{T}[x_2(n_1, n_2)] = y_2(n_1, n_2)$, a and b are any scalar constants, and $A \leftrightarrow B$ means that A implies B and B implies A. The condition in Eq. (7.12) is called the principle of *superposition*.

The shift invariance of a system is defined as

$$\text{Shift invariance} \quad \longleftrightarrow \quad \mathrm{T}[x(n_1 - m_1, n_2 - m_2)] = y(n_1 - m_1, n_2 - m_2) \tag{7.13}$$

where $y(n_1, n_2) = \mathrm{T}[x(n_1, n_2)]$ and m_1 and m_2 are any integers.

For a linear and shift-invariant system, we can derive the following input-output relation using Eqs. (7.2), (7.12), and (7.13):

$$y(n_1, n_2) = \mathrm{T}[x(n_1, n_2)] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) \cdot h(n_1 - k_1, n_2 - k_2) \tag{7.14}$$

where $h(n_1, n_2) = \mathrm{T}[\delta(n_1, n_2)]$, the response of the system when the input is $\delta(n_1, n_2)$. Equation (7.14) states that the unit sample response of a linear shift-invariant system is completely characterized by the unit sample response $h(n_1, n_2)$. Specifically, for a linear shift-invariant system, knowledge of $h(n_1, n_2)$ alone allows us to determine the output of the system to any input from Eq. (7.14). Equation (7.14) is referred to as *convolution* and is denoted by the convolution operator $*$. For a linear shift-invariant system,

$$\begin{aligned} y(n_1, n_2) &= x(n_1, n_2) * h(n_1, n_2) \\ &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) \cdot h(n_1 - k_1, n_2 - k_2) \end{aligned} \tag{7.15}$$

Note that the unit sample response $h(n_1, n_2)$, which plays such an important role for a linear shift-invariant system, loses its significance for a nonlinear or shift-variant system. All the results in this section are straightforward extensions of 1-D results.

7.1.3 Convolution

The convolution operator in Eq. (7.15) has a number of properties that are straightforward extensions of 1-D results. Some of the more important ones are listed below.

Commutativity:

$$x(n_1, n_2) * y(n_1, n_2) = y(n_1, n_2) * x(n_1, n_2) \tag{7.16}$$

Associativity:

$$[x(n_1,n_2) * y(n_1,n_2)] * z(n_1,n_2) = x(n_1,n_2) * [y(n_1,n_2) * z(n_1,n_2)] \tag{7.17}$$

Distributivity:

$$\begin{aligned} x(n_1,n_2) * [y(n_1,n_2) + z(n_1,n_2)] \\ = [x(n_1,n_2) * y(n_1,n_2)] + [x(n_1,n_2) * z(n_1,n_2)] \end{aligned} \tag{7.18}$$

Convolution with Delayed Unit Sample Sequence:

$$x(n_1,n_2) * \delta(n_1 - m_1, n_2 - m_2) = x(n_1 - m_1, n_2 - m_2) \tag{7.19}$$

The convolution of two sequences $x(n_1,n_2)$ and $h(n_1,n_2)$ can be obtained by explicitly evaluating Eq. (7.15). It is often simpler and more instructive, however, to evaluate Eq. (7.15) graphically. Specifically, the convolution sum in Eq. (7.15) can be interpreted as multiplying two sequences $x(k_1,k_2)$ and $h(n_1 - k_1, n_2 - k_2)$ that are functions of the variables (k_1,k_2) and summing the product over all integer values of (k_1,k_2). The result, which is a function of (n_1,n_2), is the result of convolving $x(n_1,n_2)$ and $h(n_1,n_2)$. As an example, consider the two sequences $x(n_1,n_2)$ and $h(n_1,n_2)$, shown in Figs. 7.6(a) and (b). From $x(n_1,n_2)$ and $h(n_1,n_2)$, we can obtain $x(k_1,k_2)$ and $h(n_1 - k_1, n_2 - k_2)$ as functions of k_1 and k_2, as shown in Figs. 7.6(c) and (d). Note that we can obtain $h(n_1 - k_1, n_2 - k_2)$ as a function of k_1 and k_2 from $h(n_1,n_2)$ by first changing the variables n_1 and n_2 to k_1 and k_2, flipping the sequence with respect to the origin, and then shifting the result in the positive k_1 and k_2 directions by n_1 and n_2 points, respectively. Once we have obtained $x(k_1,k_2)$ and $h(k_1 - n_1, k_2 - n_2)$, we can multiply and sum them for each different set of n_1 and n_2. The result is shown in Fig. 7.6(e).

A linear shift-invariant system is called separable, if its unit sample response $h(n_1,n_2)$ is a separable sequence. For a separable system, it is possible to reduce the number of arithmetic operations in computing the convolution sum. For this reason, separable systems are sometimes used in processing images. To illustrate this, we consider an input sequence $x(n_1,n_2)$ of size $N \times N$ points and a unit sample response $h(n_1,n_2)$ of size $M \times M$ points, as follows:

$$\begin{aligned} x(n_1,n_2) &= 0 \text{ outside } 0 \le n_1 \le N-1,\ 0 \le n_2 \le N-1, \\ h(n_1,n_2) &= 0 \text{ outside } 0 \le n_1 \le M-1,\ 0 \le n_2 \le M-1 \end{aligned} \tag{7.20}$$

where $N >> M$ in typical cases. The size of nonzero values of $x(n_1,n_2)$ and $h(n_1,n_2)$ is shown in Figs. 7.7(a) and (b). Denoting the output of the system with $y(n_1,n_2)$, $y(n_1,n_2)$ can be expressed as

$$\begin{aligned} y(n_1,n_2) &= x(n_1,n_2) * h(n_1,n_2) \\ &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty} x(k_1,k_2) \cdot h(n_1 - k_1, n_2 - k_2) \end{aligned} \tag{7.21}$$

The size of nonzero values of $y(n_1,n_2)$ is shown in Fig. 7.7(c).

If Eq. (7.21) is used directly to compute $y(n_1,n_2)$, approximately $(N + M - 1)^2 \cdot M^2$ arithmetic operations (defined as one multiplication and one

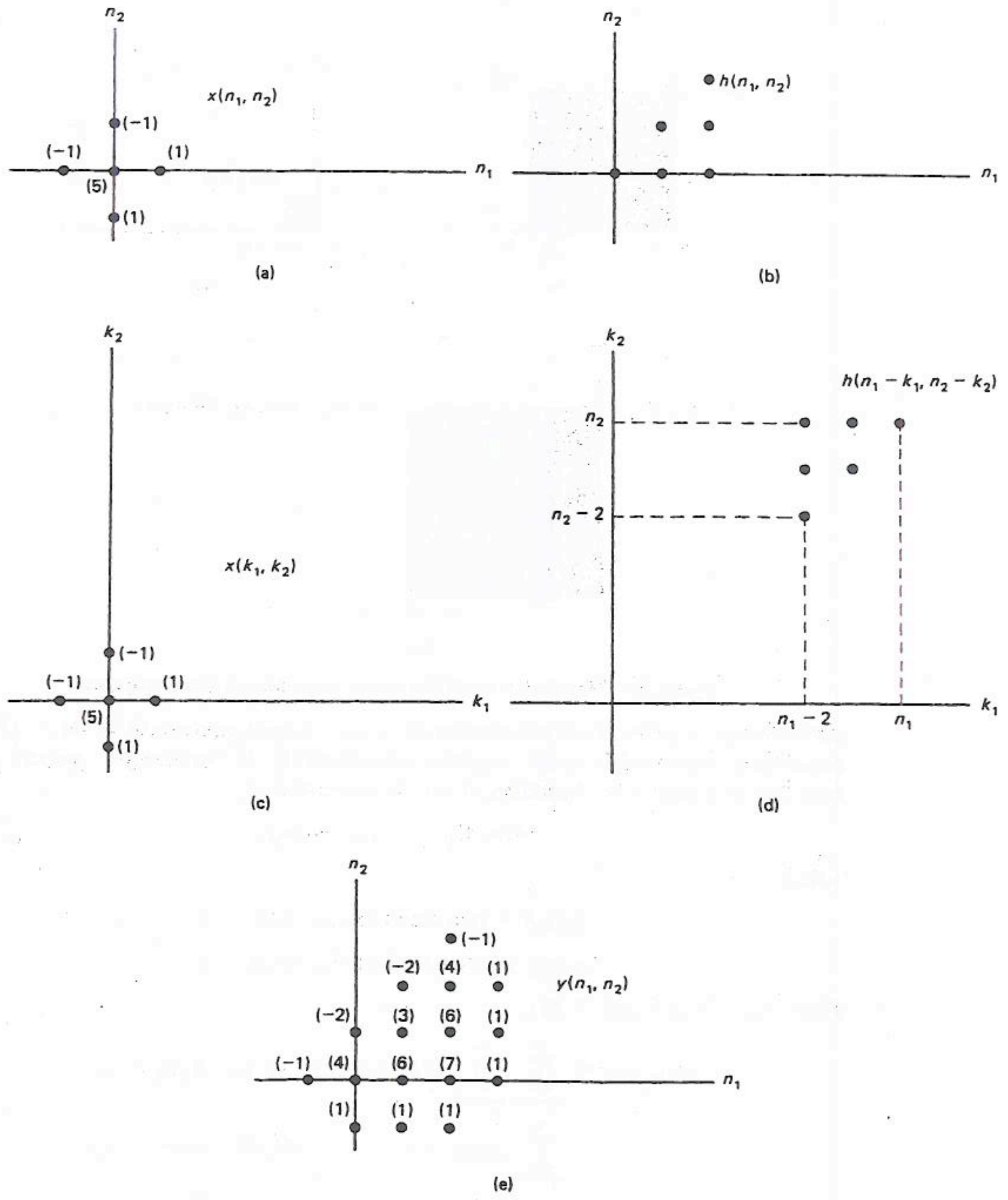


Figure 7.6 An example of linear convolution of two 2-D sequences.

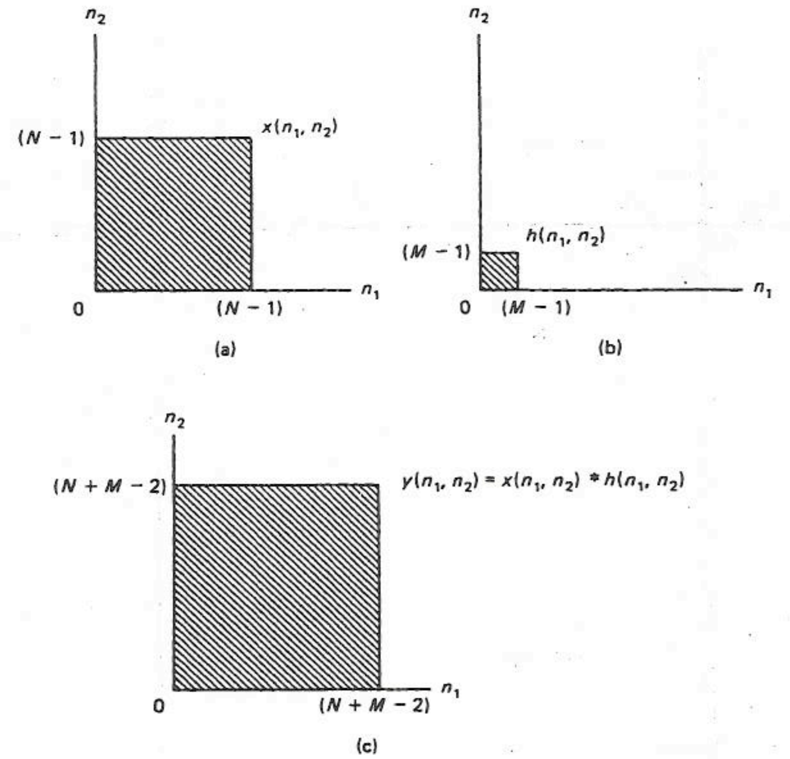


Figure 7.7 Regions for which the signal values have nonzero amplitude.

addition) are required since the number of nonzero output points is $(N + M - 1)^2$ and computing each output point requires approximately M^2 arithmetic operations. If $h(n_1, n_2)$ is a separable sequence, it can be expressed as

$$h(n_1, n_2) = h_1(n_1) \cdot h_2(n_2), \tag{7.22}$$

where

$$h_1(n_1) = 0 \text{ outside } 0 \le n_1 \le N - 1$$
$$h_2(n_2) = 0 \text{ outside } 0 \le n_2 \le M - 1$$

From Eqs. (7.21) and (7.22),

$$\begin{aligned} y(n_1, n_2) &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) \cdot h_1(n_1 - k_1) \cdot h_2(n_2 - k_2) \\ &= \sum_{k_1=-\infty}^{\infty} h_1(n_1 - k_1) \cdot \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) \cdot h_2(n_2 - k_2) \end{aligned} \tag{7.23}$$

For a fixed k_1, $\sum_{k_2=-\infty}^{\infty} x(k_1, k_2) \cdot h_2(n_2 - k_2)$ in Eq. (7.23) corresponds to a 1-D convolution of $x(k_1, n_2)$ and $h_2(n_2)$. For example, using the notation

$$f(k_1, n_2) = \sum_{k_2=-\infty}^{\infty} x(k_1, k_2) \cdot h_2(n_2 - k_2), \tag{7.24}$$

$f(0, n_2)$ is the 1-D convolution of $x(0, n_2)$ with $h_2(n_2)$. Since there are N different values of k_1 for which $x(k_1, k_2)$ is nonzero, computing $f(k_1, n_2)$ requires N 1-D convolutions and therefore requires approximately $N \cdot (N + M - 1) \cdot M$ arithmetic operations. Once $f(k_1, n_2)$ is computed, $y(n_1, n_2)$ can be computed from Eqs. (7.23) and (7.24) as

$$y(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} h_1(n_1 - k_1) \cdot f(k_1, n_2) \tag{7.25}$$

For a fixed n_2, Eq. (7.25) is a 1-D convolution of $h_1(n_1)$ and $f(n_1, n_2)$. For example, $y(n_1, 1)$ is a 1-D convolution of $f(n_1, 1)$ and $h_1(n_1)$. Since there are $(N + M - 1)$ different values of n_2 for which $f(k_1, n_2)$ is nonzero, computing $y(n_1, n_2)$ requires $(N + M - 1)$ 1-D convolutions and therefore requires approximately $(N + M - 1)^2 \cdot M$ arithmetic operations. Assuming $M \ll N$, the total number of arithmetic operations required is approximately $(N + M - 1)^2 \cdot 2M$, which compares favorably with $(N + M - 1)^2 \cdot M^2$. When $M = 10$, exploiting the separability of $h(n_1, n_2)$ reduces the number of arithmetic operations by approximately a factor of 5.

7.1.4 Stable Systems and Special Support Systems

For practical considerations, it is often appropriate to impose additional constraints on the class of systems we consider. Systems with those constraints are stable systems and special support systems.

A system is considered *stable* in the bounded input–bounded output (BIBO) sense if and only if a bounded input always leads to a bounded output. Stability is often a desirable constraint to impose since an unstable system can generate an unbounded output, which can cause system overload or other difficulties. From this definition and Eq. (7.15), it can be shown that a necessary and sufficient condition for a linear shift-invariant system to be stable is that its unit sample response $h(n_1, n_2)$ be absolutely summable. For a linear shift-invariant system,

$$\text{Stability} \quad \longleftrightarrow \quad \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} |h(n_1,n_2)| < \infty \tag{7.26}$$

Even though Eq. (7.26) is a straightforward extension of 1-D results, issues related to stability, such as testing the stability of a system, are quite different between 1-D and 2-D results, as we will explore further in Section 7.5. Because of Eq. (7.26), an absolutely summable sequence is defined to be a stable sequence.

Special support systems can be viewed as extensions of 1-D causal systems. Specifically, a 1-D system is *causal* if and only if the current output $y(n)$ does not depend on any future values of input, e.g., $x(n + 1), x(n + 2), x(n + 3), \ldots$. With this definition, it can be shown that a necessary and sufficient condition for a 1-D linear shift-invariant system to be causal is that its unit sample response $h(n)$ be zero for $n < 0$. Causality is often a desirable constraint to impose in designing 1-D systems. In typical 1-D signal processing applications such as speech processing, there is a well-defined time reference, and a noncausal system requires delay, which is undesirable in many real-time applications. In typical 2-D signal processing applications such as image processing, the causality may not be necessary. At a given time, a whole image may be available for processing, and it may be processed from left to right, from top to bottom, or in any direction. Even though the notion of causality may not be as important a constraint in 2-D signal processing, it is useful to extend the notion that a 1-D causal linear shift-invariant system has its unit sample response $h(n)$ whose nonzero values lie in a particular region. A 2-D linear shift-invariant system whose unit sample response $h(n_1, n_2)$ has all its nonzero values in a particular region is called a *special support system*.

A 2-D linear shift-invariant system with its unit sample response $h(n_1, n_2)$ is called a *quadrant support system* when $h(n_1, n_2)$ is a quadrant support sequence. A sequence is called a *quadrant support sequence*, or quadrant sequence, when all its nonzero values lie in one quadrant. An example of a first-quadrant sequence is the unit step sequence $u(n_1, n_2)$.

A 2-D linear shift-invariant system with its unit sample response $h(n_1, n_2)$ is called a *wedge support system* when $h(n_1, n_2)$ is a wedge support sequence. Consider two lines emanating from the origin. If all the nonzero values in a sequence lie in the region bounded by the two lines with an angle less than 180° between the two lines, the sequence is called a *wedge support sequence*, or wedge sequence. An example of a wedge sequence is shown in Fig. 7.8.

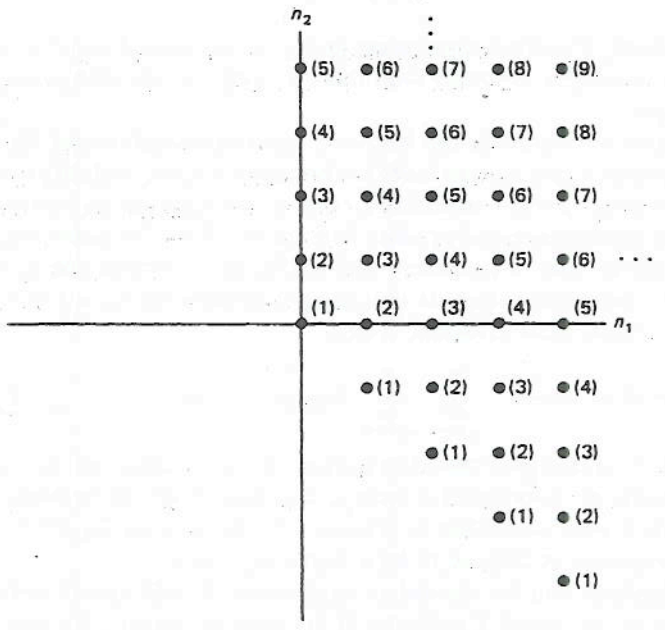


Figure 7.8 An example of a wedge sequence.

Quadrant sequences and wedge sequences are closely related. A quadrant sequence is always a wedge sequence. In addition, it can be shown that any wedge sequence can always be transformed into a first-quadrant sequence by a linear change of variables. To illustrate this, we consider the wedge sequence $x(n_1, n_2)$ shown in Fig. 7.8. Suppose we obtain a new sequence $y(n_1, n_2)$ from $x(n_1, n_2)$ by the following linear change of variables:

$$y(n_1, n_2) = x(m_1, m_2)\big|_{m_1=n_1,\, m_2=n_2-n_1} \tag{7.27}$$

The sequence $y(n_1, n_2)$ we obtained is shown in Fig. 7.9, and it is clearly a first-quadrant sequence. For this example, the stability of $x(n_1, n_2)$ is equivalent to the stability of $y(n_1, n_2)$, since $\sum_{n_1} \sum_{n_2} |x(n_1, n_2)| = \sum_{n_1} \sum_{n_2} |y(n_1, n_2)|$. It is possible to show that a proper choice of linear change of variables maps any wedge sequence to a first-quadrant sequence without affecting the stability.

The notions that a wedge sequence can always be transformed to a first-quadrant sequence by a simple linear mapping of variables and that the stability of these two sequences is equivalent with a proper choice of linear mapping are very useful in discussing the stability of a 2-D system. As we discuss later, our primary concern in checking the stability of a 2-D system will be limited to a class of systems known as "recursively computable" systems. For a recursively computable system, the stability depends on the stability of a wedge sequence $h(n_1, n_2)$. Our approach to checking the stability of a wedge sequence $h(n_1, n_2)$ will be to transform $h(n_1, n_2)$ to a first-quadrant sequence $h'(n_1, n_2)$ by a linear transformation of variables and then to check the stability of $h'(n_1, n_2)$. It is much easier to develop stability theorems that apply to first-quadrant sequences than to wedge sequences. This is discussed further in Section 7.5.

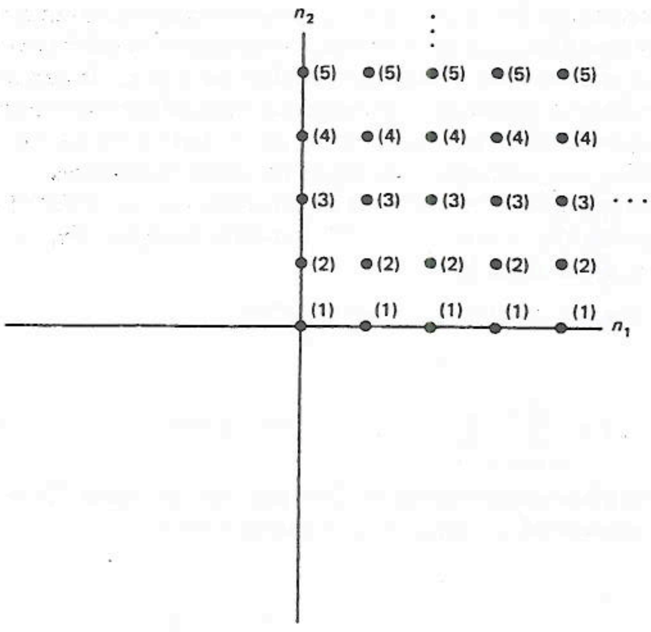


Figure 7.9 First-quadrant sequence obtained by linear mapping of variables of the wedge sequence in Fig. 7.8.

7.2 FOURIER TRANSFORM

7.2.1 Fourier Transform Pair

It is remarkable that any stable sequence $x(n_1, n_2)$ can be obtained by appropriately combining complex exponentials of the form $X(\omega_1, \omega_2) \cdot e^{j\omega_1 n_1} \cdot e^{j\omega_2 n_2}$. The function $X(\omega_1, \omega_2)$ that represents the amplitude associated with the complex exponential

$e^{j\omega_1 n_1} \cdot e^{j\omega_2 n_2}$ can be obtained from $x(n_1, n_2)$. The relationships between $x(n_1, n_2)$ and $X(\omega_1, \omega_2)$ are given by

Discrete Space Fourier Transform Pair

$$X(\omega_1, \omega_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) \cdot e^{-j\omega_1 n_1} \cdot e^{-j\omega_2 n_2} \tag{7.28}$$

$$x(n_1, n_2) = \frac{1}{(2\pi)^2} \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} X(\omega_1, \omega_2) \cdot e^{j\omega_1 n_1} \cdot e^{j\omega_2 n_2} \tag{7.29}$$

Equation (7.28) shows how the amplitude $X(\omega_1, \omega_2)$ associated with the exponential $e^{j\omega_1 n_1} \cdot e^{j\omega_2 n_2}$ can be determined from $x(n_1, n_2)$. The function $X(\omega_1, \omega_2)$ is called the *discrete space Fourier transform*, or Fourier transform, of $x(n_1, n_2)$. The sequence $x(n_1, n_2)$ is called the *inverse discrete space Fourier transform*, or inverse Fourier transform, of $X(\omega_1, \omega_2)$. The consistency of Eqs. (7.28) and (7.29) can be easily shown by combining them.

From Eq. (7.28), we can see that $X(\omega_1, \omega_2)$ is in general complex, even though $x(n_1, n_2)$ may be real, and that $X(\omega_1, \omega_2)$ is a function of continuous variables ω_1 and ω_2, even though $x(n_1, n_2)$ is a function of discrete variables n_1 and n_2. In addition, $X(\omega_1, \omega_2)$ is always periodic with period 2π with respect to each of the two variables ω_1 and ω_2; i.e., $X(\omega_1, \omega_2) = X(\omega_1 + 2\pi, \omega_2) = X(\omega_1, \omega_2 + 2\pi)$ for all ω_1 and ω_2. We can also show that $X(\omega_1, \omega_2)$ uniformly converges for stable sequences.

The 2-D complex exponential $e^{j\omega_1 n_1} \cdot e^{j\omega_2 n_2}$ is an eigenfunction of a 2-D linear shift-invariant system. Specifically, when $e^{j\omega_1 n_1} \cdot e^{j\omega_2 n_2}$ is used as an input $x(n_1, n_2)$, the output of the system $y(n_1, n_2)$ is given by

$$y(n_1, n_2) = H(\omega_1, \omega_2) \cdot e^{j\omega_1 n_1} \cdot e^{j\omega_2 n_2}$$

where

$$H(\omega_1, \omega_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} h(k_1, k_2) \cdot e^{-j\omega_1 k_1} \cdot e^{-j\omega_2 k_2} \tag{7.30}$$

From Eq. (7.30), the output is a scaled version of the input, and the scalar $H(\omega_1, \omega_2)$ is called the *frequency response* of the linear shift-invariant system.

7.2.2 Properties

We can derive a number of useful properties from the Fourier transform pair in Eqs. (7.28) and (7.29). Some of the more important properties, often useful in practice, are listed in Table 7.1. Most of these properties are essentially straightforward extensions of 1-D Fourier transform properties. The only exception is property 4, which applies to separable sequences. If a 2-D sequence $x(n_1, n_2)$ can be written as $x_1(n_1) \cdot x_2(n_2)$, then its Fourier transform $X(\omega_1, \omega_2)$ is given by $X_1(\omega_1) \cdot X_2(\omega_2)$, where $X_1(\omega_1)$ and $X_2(\omega_2)$ represent the 1-D Fourier transforms of $x_1(n_1)$ and $x_2(n_2)$ respectively. This property follows directly from the Fourier transform pair of equations (7.28) and (7.29).

TABLE 7.1 PROPERTIES OF THE FOURIER TRANSFORM

$x(n_1, n_2) \longleftrightarrow X(\omega_1, \omega_2) \qquad y(n_1, n_2) \longleftrightarrow Y(\omega_1, \omega_2)$

Property 1: Linearity

$$a \cdot x(n_1, n_2) + b \cdot y(n_1, n_2) \longleftrightarrow a \cdot X(\omega_1, \omega_2) + b \cdot Y(\omega_1, \omega_2)$$

Property 2: Convolution

$$x(n_1, n_2) * y(n_1, n_2) \longleftrightarrow X(\omega_1, \omega_2) \cdot Y(\omega_1, \omega_2)$$

Property 3: Modulation

$$x(n_1, n_2) y(n_1, n_2) \longleftrightarrow X(\omega_1, \omega_2) \circledast Y(\omega_1, \omega_2)$$
$$= \frac{1}{4\pi^2} \cdot \int_{\theta_1=-\pi}^{\pi} \int_{\theta_2=-\pi}^{\pi} X(\theta_1, \theta_2) \cdot Y(\omega_1 - \theta_1, \omega_2 - \theta_2) \cdot d\theta_1 \cdot d\theta_2$$

Property 4: Separable sequence

$$x_1(n_1) x_2(n_2) \longleftrightarrow X_1(\omega_1) X_2(\omega_2)$$

Property 5: Shift of a sequence or Fourier transform

(a) $x(n_1 - m_1, n_2 - m_2) \longleftrightarrow X(\omega_1, \omega_2) \cdot e^{-j\omega_1 m_1} \cdot e^{-j\omega_2 m_2}$

(b) $e^{j\nu_1 n_1} \cdot e^{j\nu_2 n_2} \cdot x(n_1, n_2) \longleftrightarrow X(\omega_1 - \nu_1, \omega_2 - \nu_2)$

Property 6: Differentiation

(a) $-jn_1 \cdot x(n_1, n_2) \longleftrightarrow \dfrac{\partial X(\omega_1, \omega_2)}{\partial \omega_1}$

(b) $-jn_2 \cdot x(n_1, n_2) \longleftrightarrow \dfrac{\partial X(\omega_1, \omega_2)}{\partial \omega_2}$

Property 7: Parseval's theorem

$$\sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} |x(n_1, n_2)|^2 = \frac{1}{(2\pi)^2} \cdot \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} |X(\omega_1, \omega_2)|^2 \cdot d\omega_1 \cdot d\omega_2$$

Property 8: Symmetry properties

(a) $x(n_1, n_2)$: real $\longleftrightarrow X(\omega_1, \omega_2) = X^*(-\omega_1, -\omega_2)$
$X_R(\omega_1, \omega_2), |X(\omega_1, \omega_2)|$: even (symmetric with respect to the origin)
$X_I(\omega_1, \omega_2), \theta_x(\omega_1, \omega_2)$: odd (antisymmetric with respect to the origin)

(b) $x(n_1, n_2)$: real and even $\longleftrightarrow X(\omega_1, \omega_2)$: real and even

(c) $x(n_1, n_2)$: real and odd $\longleftrightarrow X(\omega_1, \omega_2)$: pure imaginary and odd

7.2.3 Examples

Example 1

We wish to determine $H(\omega_1, \omega_2)$ for the sequence $h(n_1, n_2)$ shown in Fig. 7.10. From Eq. (7.28),

$$H(\omega_1, \omega_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} h(n_1, n_2) \cdot e^{-j\omega_1 n_1} \cdot e^{-j\omega_2 n_2}$$
$$= \frac{1}{3} + \frac{1}{6} \cdot e^{-j\omega_1} + \frac{1}{6} \cdot e^{-j\omega_2} + \frac{1}{6} \cdot e^{j\omega_1} + \frac{1}{6} \cdot e^{j\omega_2}$$
$$= \frac{1}{3} + \frac{1}{3} \cos \omega_1 + \frac{1}{3} \cos \omega_2$$

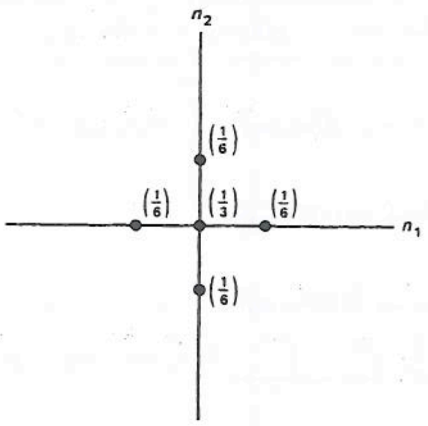

n_2
$\left(\frac{1}{6}\right)$
$\left(\frac{1}{6}\right)$
$\left(\frac{1}{3}\right)$
$\left(\frac{1}{6}\right)$
n_1
$\left(\frac{1}{6}\right)$

Figure 7.10 A 2-D sequence $h(n_1, n_2)$.

The function $H(\omega_1, \omega_2)$ for this example is real, and $|H(\omega_1, \omega_2)|$ is shown in Fig. 7.11. If $H(\omega_1, \omega_2)$ were the frequency response of a linear shift-invariant system, the system would correspond to a lowpass filter. The function $H(\omega_1, \omega_2)$ has smaller values in frequency regions away from the origin. A lowpass filter when applied to an image blurs the image. Figure 7.12(a) shows an image of 512 × 512 pixels. Figure 7.12(b) shows the image obtained by processing the image in Fig. 7.12(a) with a lowpass filter whose unit sample response $h(n_1, n_2)$ is shown in Fig. 7.10.

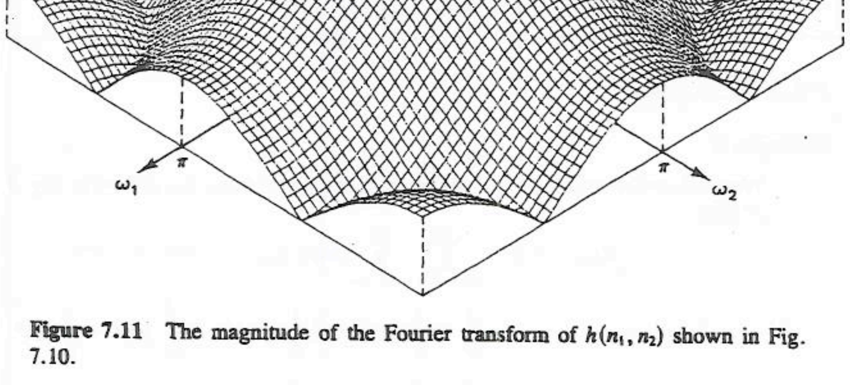
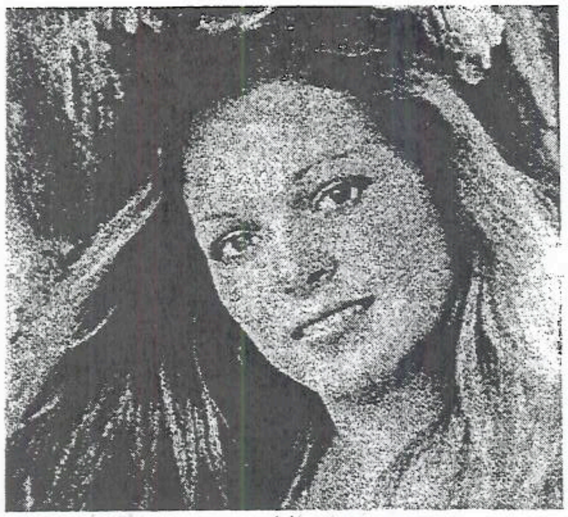


Figure 7.11 The magnitude of the Fourier transform of $h(n_1, n_2)$ shown in Fig. 7.10.


(a)

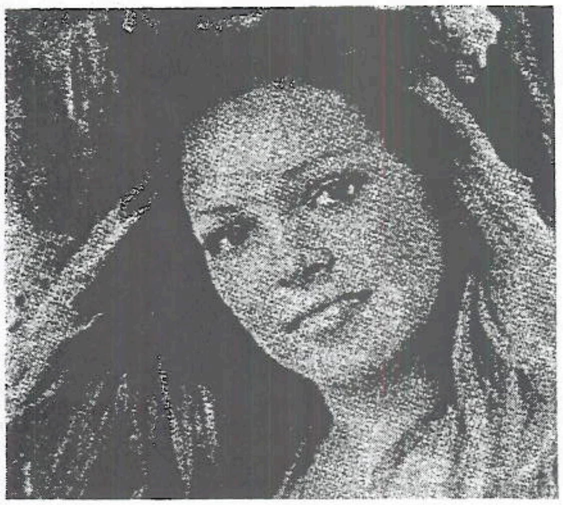

(b)

Figure 7.12 (a) An original image of 256 × 256 pixels; (b) the image in part (a) filtered by the lowpass filter shown in Fig. 7.11.

Example 2

We wish to determine $h(n_1, n_2)$ for the Fourier transform $H(\omega_1, \omega_2)$ shown in Fig. 7.13. The function $H(\omega_1, \omega_2)$ is given by

$$H(\omega_1, \omega_2) = \begin{cases} 1, & |\omega_1| \le a \text{ and } |\omega_2| \le b \quad \text{(shaded region)} \\ 0, & a < |\omega_1| \le \pi \text{ or } b < |\omega_2| \le \pi \quad \text{(unshaded region)} \end{cases}$$

Since $H(\omega_1, \omega_2)$ is always periodic with period 2π along each of the variables ω_1 and ω_2, $H(\omega_1, \omega_2)$ is shown only for $|\omega_1| \le \pi$ and $|\omega_2| \le \pi$. The function $H(\omega_1, \omega_2)$ can be expressed as $H_1(\omega_1) \cdot H_2(\omega_2)$, where one possible choice of $H_1(\omega_1)$ and $H_2(\omega_2)$ is also shown in Fig. 7.13. Computing the 1-D inverse Fourier transforms of $H_1(\omega_1)$ and $H_2(\omega_2)$

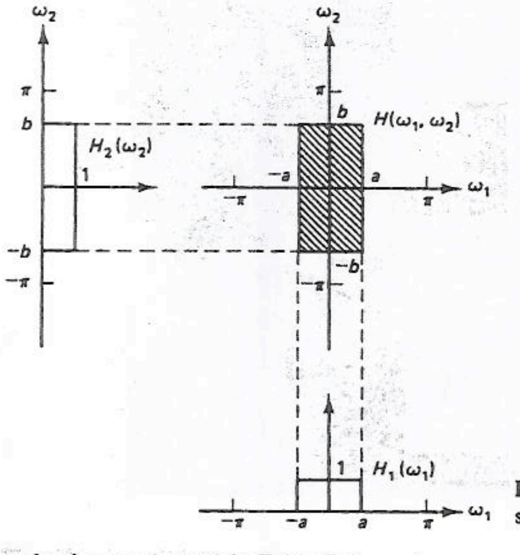


Figure 7.13 Frequency response of a separable ideal lowpass filter.

and using property 4 in Table 7.1, we have

$$h(n_1, n_2) = h_1(n_1) \cdot h_2(n_2) = \frac{\sin an_1}{\pi n_1} \cdot \frac{\sin bn_2}{\pi n_2}$$

Example 3

We wish to determine $h(n_1, n_2)$ for the Fourier transform $H(\omega_1, \omega_2)$ shown in Fig. 7.14. The function $H(\omega_1, \omega_2)$ is given by

$$H(\omega_1, \omega_2) = \begin{cases} 1, & \sqrt{\omega_1^2 + \omega_2^2} \le \omega_c \quad \text{(shaded region)} \\ 0, & \sqrt{\omega_1^2 + \omega_2^2} > \omega_c \text{ and } |\omega_1|, |\omega_2| < \pi \quad \text{(unshaded region)} \end{cases}$$

If $H(\omega_1, \omega_2)$ above is the frequency response of a 2-D linear shift-invariant system, the system is called a *circularly symmetric ideal lowpass filter*. The inverse Fourier transform of $H(\omega_1, \omega_2)$ in this example requires a fair amount of algebra, and the result is

$$h(n_1, n_2) = \frac{\omega_c}{2\pi\sqrt{n_1^2 + n_2^2}} \cdot J_1\left(\omega_c \cdot \sqrt{n_1^2 + n_2^2}\right) \tag{7.31}$$

where $J_1(x)$ represents the Bessel function of the first kind and the first order. This example shows that 2-D Fourier transform or inverse Fourier transform operations can

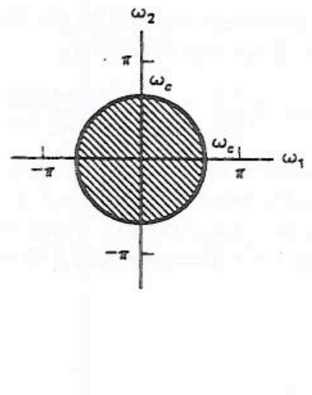


Figure 7.14 Frequency response of a circularly symmetric ideal lowpass filter.

become quite complex algebraically relative to 1-D Fourier transform or inverse Fourier transform operations, even though the 2-D Fourier transform pair and many 2-D Fourier transform properties are straightforward extensions of 1-D results. From Eq. (7.31), we see that the unit sample response of a 2-D circularly symmetric ideal lowpass filter is also circularly symmetric, e.g., it is a function of $n_1^2 + n_2^2$. This is a special case of the result that circular symmetry of $H(\omega_1, \omega_2)$ implies circular symmetry of $h(n_1, n_2)$. We note, however, that circular symmetry of $h(n_1, n_2)$ does not imply circular symmetry of $H(\omega_1, \omega_2)$. The function $J_1(x)/x$ is sketched in Fig. 7.15. The sequence $h(n_1, n_2)$ in Eq. (7.31) is sketched in Fig. 7.16 for the case $\omega_c = 0.4\pi$.

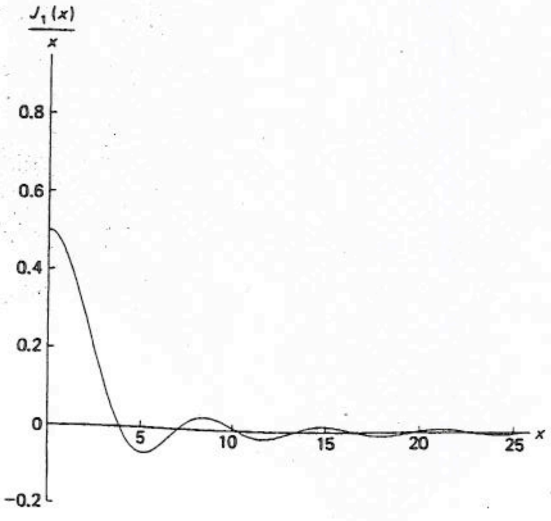


Figure 7.15 Sketch of $J_1(x)/x$, where $J_1(x)$ is the Bessel function of the first kind and the first order.

7.3 z-TRANSFORM

7.3.1 z-Transform

The Fourier transform discussed in Section 7.2 uniformly converges for stable sequences, and many interesting classes of unstable sequences, such as the unit step sequence $u(n_1, n_2)$, cannot be represented by their Fourier transforms. In this section, we discuss the z-transform representation of a sequence, which converges for a much wider class of signals.

The z-transform of a sequence $x(n_1, n_2)$ is denoted by $X(z_1, z_2)$ and is defined by

$$X(z_1, z_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) \cdot z_1^{-n_1} \cdot z_2^{-n_2} \tag{7.32}$$

where z_1 and z_2 are complex variables. Since each of the variables z_1 and z_2 represents 2-D space, the space represented by (z_1, z_2) is four-dimensional (4-D). As a result, it is extremely difficult to visualize points or segments of $X(z_1, z_2)$.

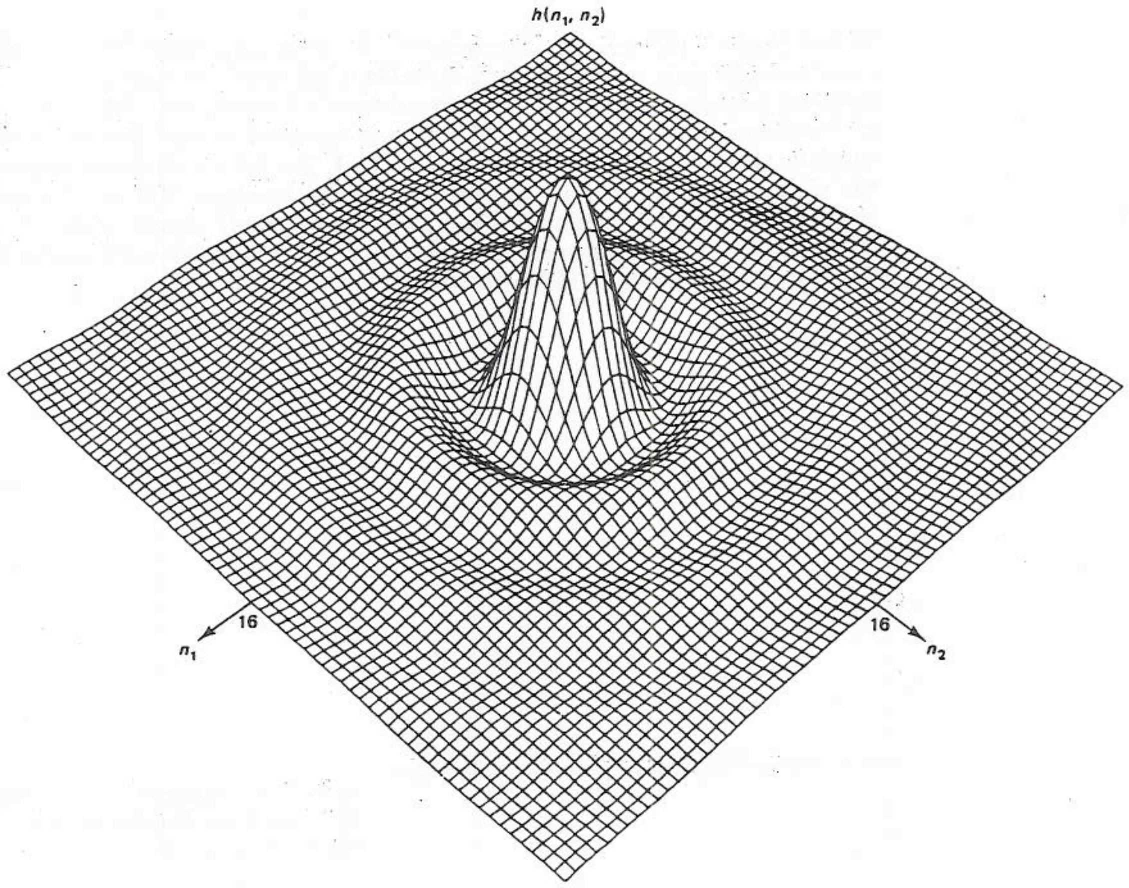


Figure 7.16 The unit sample response of a circularly symmetric ideal lowpass filter with $\omega_c = 0.4\pi$ in Eq. (7.31). The value at the origin, $h(0, 0)$, is 0.126.

From Eqs. (7.28) and (7.32), we see that $X(z_1, z_2)$ is related to $X(\omega_1, \omega_2)$ by

$$X(z_1, z_2)\big|_{z_1=e^{j\omega_1}, z_2=e^{j\omega_2}} = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) \cdot e^{-j\omega_1 n_1} \cdot e^{-j\omega_2 n_2} = X(\omega_1, \omega_2) \tag{7.33}$$

Equation (7.33) states that $X(\omega_1, \omega_2)$ is $X(z_1, z_2)$ evaluated at $z_1 = e^{j\omega_1}$ and $z_2 = e^{j\omega_2}$. This is one reason why the z-transform is considered a generalization of the Fourier transform. The 2-D space represented by $(z_1 = e^{j\omega_1}, z_2 = e^{j\omega_2})$ is called the *unit surface*.

Suppose $X(z_1, z_2)$ in Eq. (7.32) is evaluated along $(z_1 = r_1 \cdot e^{j\omega_1}, z_2 = r_2 \cdot e^{j\omega_2})$, where r_1 and ω_1 are the radius and argument in the z_1-plane and r_2 and ω_2 are the radius and argument in the z_2-plane. The function $X(z_1, z_2)$ can be expressed as

$$\begin{aligned} X(z_1, z_2)\big|_{z_1=r_1 \cdot e^{j\omega_1}, z_2=r_2 \cdot e^{j\omega_2}} &= \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) \cdot r_1^{-n_1} \cdot r_2^{-n_2} \cdot e^{-j\omega_1 n_1} \cdot e^{-j\omega_2 n_2} \\ &= F[x(n_1, n_2) \cdot r_1^{-n_1} \cdot r_2^{-n_2}] \end{aligned} \tag{7.34}$$

where $F[x(n_1, n_2) \cdot r_1^{-n_1} \cdot r_2^{-n_2}]$ represents the Fourier transform of $x(n_1, n_2) \cdot r_1^{-n_1} \cdot r_2^{-n_2}$. Since $X(\omega_1, \omega_2)$ uniformly converges for an absolutely summable sequence, from Eq. (7.34) $X(z_1, z_2)$ uniformly converges when $r_1^{-n_1} \cdot r_2^{-n_2} \cdot x(n_1, n_2)$ is absolutely summable:

$$\sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} |r_1^{-n_1} \cdot r_2^{-n_2} \cdot x(n_1, n_2)| < \infty, \qquad \text{where } r_1 = |z_1|,\ r_2 = |z_2| \tag{7.35}$$

From Eq. (7.35), the convergence of $X(z_1, z_2)$ will generally depend on the value of $r_1 = |z_1|$ and $r_2 = |z_2|$. For example, for the unit step sequence $u(n_1, n_2)$, $r_1^{-n_1} \cdot r_2^{-n_2} \cdot u(n_1, n_2)$ is absolutely summable only for $(|z_1| > 1, |z_2| > 1)$, and its z-transform converges only for $(|z_1| > 1, |z_2| > 1)$. The region in the (z_1, z_2)-plane where $X(z_1, z_2)$ uniformly converges is called the *region of convergence (ROC)*.

For 1-D signals, the ROC is typically bounded by two concentric circles whose origin is at $|z| = 0$, as shown in Fig. 7.17(a). For 2-D signals, (z_1, z_2) represents a 4-D space and therefore the ROC cannot be sketched in a way analogous to that in Fig. 7.17(a). Fortunately, however, the ROC depends only on $|z|$ for 1-D signals and on $|z_1|$ and $|z_2|$ for 2-D signals. Therefore, an alternative way to sketch the ROC for 1-D signals is to use the $|z|$-axis. The ROC in Fig. 7.17(a) sketched using the $|z|$-axis is shown in Fig. 7.17(b). For 2-D signals, we can use the $(|z_1|, |z_2|)$-axes to sketch the ROC. An example of the ROC sketched in the $(|z_1|, |z_2|)$-plane is shown in Fig. 7.17(c). In this sketch, each point in the $(|z_1|, |z_2|)$-plane corresponds to a 2-D subspace in the 4-D (z_1, z_2) space. The ROC plays an important role in the z-transform representation of a sequence, as we will see shortly.

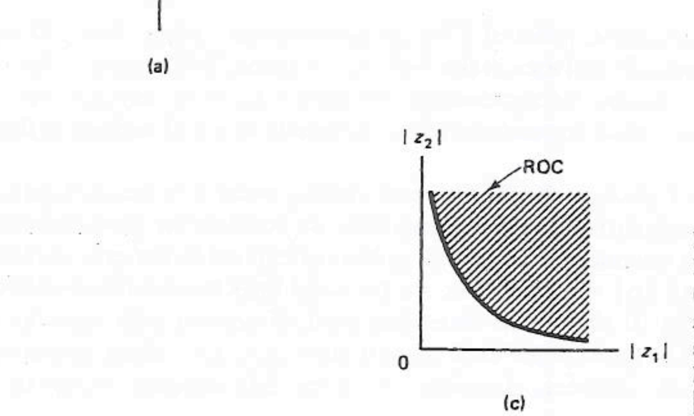


Figure 7.17 The representation of ROCs for 1-D and 2-D z-transforms. (a) A typical ROC for a 1-D z-transform; (b) the ROC in part (a) sketched using the $|z|$-axis; (c) a typical ROC for a 2-D z-transform using the $|z_1|$- and $|z_2|$-axes.

7.3.2 z-Transform Examples

Example 1

We wish to determine the z-transform and its ROC for the following sequence:

$$x(n_1, n_2) = a^{n_1} \cdot b^{n_2} \cdot u(n_1, n_2)$$

The sequence is sketched in Fig. 7.18(a). Using the z-transform definition in Eq. (7.32), we have

$$\begin{aligned} X(z_1, z_2) &= \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} a^{n_1} \cdot b^{n_2} \cdot u(n_1, n_2) \cdot z_1^{-n_1} \cdot z_2^{-n_2} \\ &= \sum_{n_1=0}^{\infty} (a \cdot z_1^{-1})^{n_1} \cdot \sum_{n_2=0}^{\infty} (b \cdot z_2^{-1})^{n_2} \\ &= \frac{1}{1 - a \cdot z_1^{-1}} \cdot \frac{1}{1 - b \cdot z_2^{-1}}, \qquad |z_1| > |a| \text{ and } |z_2| > |b| \end{aligned}$$

The ROC is sketched in Fig. 7.18(b).

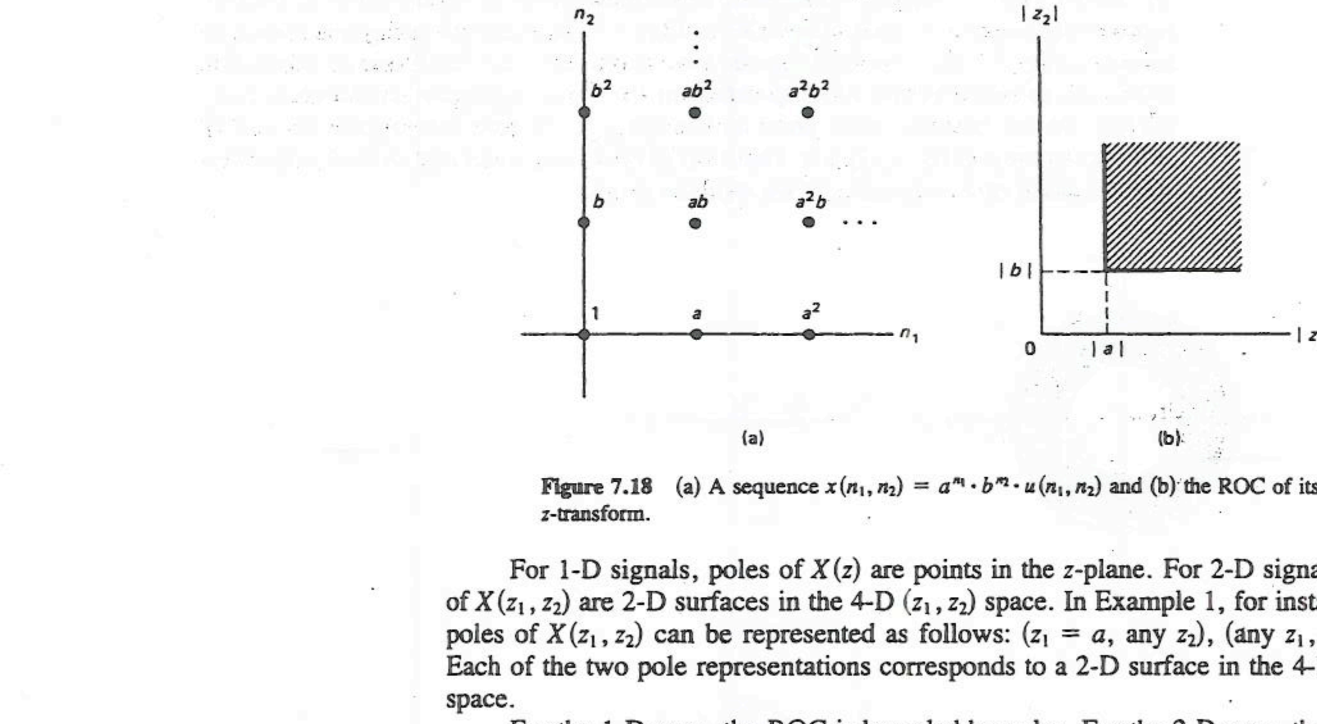


Figure 7.18 (a) A sequence $x(n_1, n_2) = a^{n_1} \cdot b^{n_2} \cdot u(n_1, n_2)$ and (b) the ROC of its z-transform.

For 1-D signals, poles of $X(z)$ are points in the z-plane. For 2-D signals, poles of $X(z_1, z_2)$ are 2-D surfaces in the 4-D (z_1, z_2) space. In Example 1, for instance, the poles of $X(z_1, z_2)$ can be represented as follows: $(z_1 = a$, any $z_2)$, (any $z_1, z_2 = b)$. Each of the two pole representations corresponds to a 2-D surface in the 4-D (z_1, z_2) space.

For the 1-D case, the ROC is bounded by poles. For the 2-D case, the ROC is bounded by pole surfaces. To illustrate this, we consider the pole surfaces in Example 1. Taking the magnitudes of z_1 and z_2 that correspond to the pole surfaces, we have $|z_1| = |a|$ and $|z_2| = |b|$. These are the solid lines that bound the ROC, as shown in Fig. 7.18(b). It should be noted that each of the two solid lines in Fig. 7.18(b) represents a 3-D space since each point in the $(|z_1|, |z_2|)$-plane corresponds to a 2-D space. The pole surfaces, therefore, lie in the 2-D subspace within the 3-D spaces corresponding to the two solid lines in Fig. 7.18(b). To see this point more clearly, consider the 3-D space corresponding to $|z_1| = |a|$. This space can be represented as $(z_1 = |a| \cdot e^{j\omega_1}$, any $z_2)$, which is shown in Fig. 7.19. The pole surface corresponding to $(z_1 = a$, any $z_2)$ is sketched in the shaded region in Fig. 7.19.

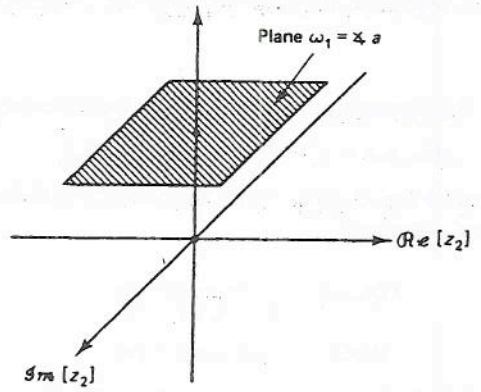


Figure 7.19 Pole surface within the 3-D space of $(|z_1| = |a|, z_2)$ for the sequence in Fig. 7.18.

Example 2

We wish to determine the z-transform and its ROC for the following sequence:

$$x(n_1, n_2) = -a^{n_1} \cdot b^{n_2} \cdot u(-n_1 - 1, n_2)$$

The sequence is sketched in Fig. 7.20(a). Using the z-transform definition in Eq. (7.32) and after a little algebra, we have

$$X(z_1, z_2) = \frac{1}{1 - a \cdot z_1^{-1}} \cdot \frac{1}{1 - b \cdot z_2^{-1}}$$

$$\text{ROC:} \quad |z_1| < |a|,\ |z_2| > |b|$$

$$\text{Pole Surfaces:} \quad (z_1 = a, \text{ any } z_2),\ (\text{any } z_1,\ z_2 = b)$$

The ROC is sketched in Fig. 7.20(b).

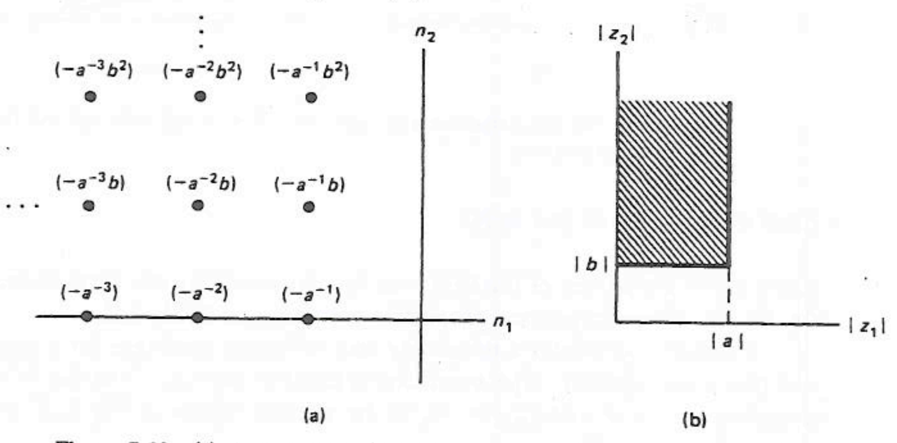


Figure 7.20 (a) A sequence $x(n_1, n_2) = -a^{n_1} \cdot b^{n_2} \cdot u(-n_1 - 1, n_2)$ and (b) the ROC of its z-transform.

Examples 1 and 2 show the importance of the ROC in the z-transform representation of a sequence. Specifically, even though the two sequences in the examples are very different, their z-transforms are exactly the same. Given only the z-transform, therefore, it is not possible to uniquely determine the sequence. The unique determination of the sequence requires not only the z-transform but its ROC.

Example 3

We wish to determine the z-transform and its ROC for the following sequence:

$$x(n_1, n_2) = a^{n_1} \cdot \delta(n_1 - n_2) \cdot u(n_1, n_2)$$

The sequence is sketched in Fig. 7.21(a). Using the z-transform definition in Eq. (7.32) and after a little algebra, we obtain

$$X(z_1, z_2) = \frac{1}{1 - a \cdot z_1^{-1} \cdot z_2^{-1}}$$

$$\text{ROC:} \quad |z_1| \cdot |z_2| > |a|$$

$$\text{Pole Surfaces:} \quad (\text{any } z_1 \neq 0,\ z_2 = a/z_1)$$

The ROC is sketched in Fig. 7.21(b). The pole surface is again a 2-D plane in the 4-D (z_1, z_2) space.

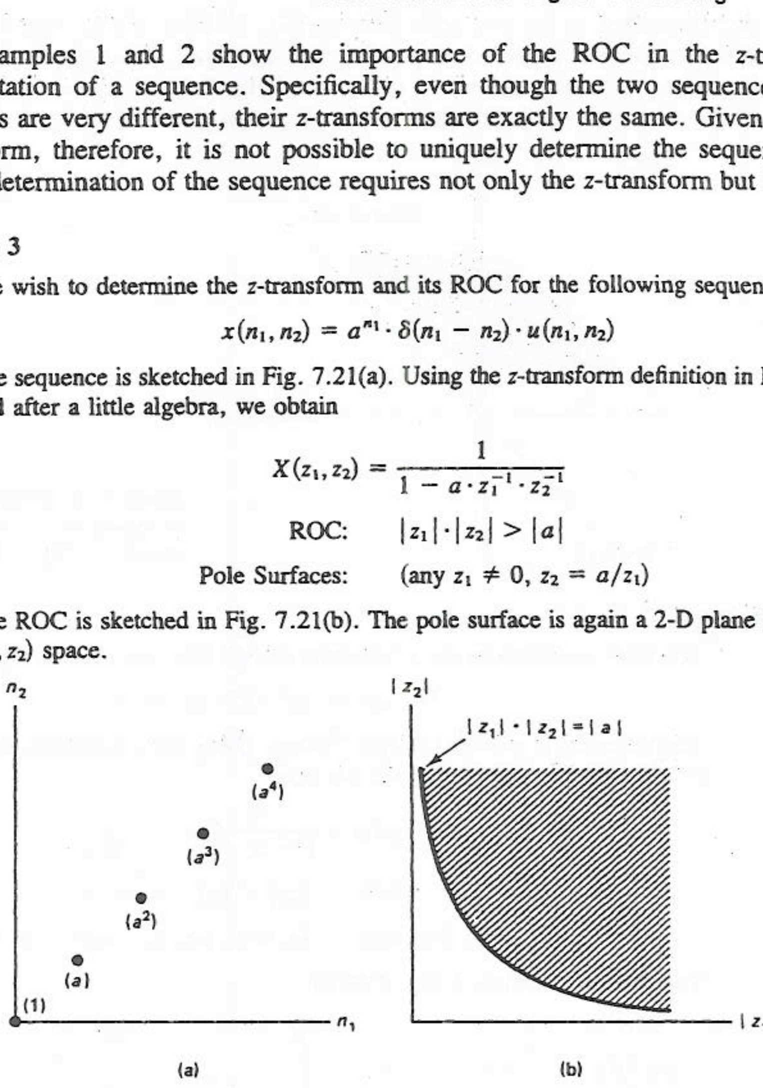


Figure 7.21 (a) The sequence $x(n_1, n_2) = a^{n_1} \cdot \delta(n_1 - n_2) \cdot u(n_1, n_2)$ and (b) the ROC of its z-transform.

7.3.3 Properties of the ROC

Many useful properties of the ROC can be obtained from the z-transform definition of Eq. (7.32). Some important properties are listed in Table 7.2.

Property 2 provides a necessary and sufficient condition for a sequence to be a first-quadrant sequence. The condition is that for any (z_1', z_2') in the ROC, all (z_1, z_2), including $|z_1| = \infty$ and $|z_2| = \infty$, in the shaded region in Fig 7.22 are also in the region of convergence.

The sketch in Fig. 7.22 is called a *constraint map* since it shows the constraints that the ROC of any first-quadrant sequence has to satisfy. Two examples of ROCs

TABLE 7.2 PROPERTIES OF THE ROC

Property 1:	A ROC is bounded by pole surfaces and is a connected region with no pole surfaces inside the ROC.
Property 2:	First-quadrant support sequence $\longleftrightarrow$ For any $(\lvert z_1' \rvert, \lvert z_2' \rvert)$ in the ROC, all $(\lvert z_1 \rvert \geq \lvert z_1' \rvert, \lvert z_2 \rvert \geq \lvert z_2' \rvert)$ are in the ROC.
Property 3:	Finite extent sequence $\longleftrightarrow$ ROC is everywhere except possibly $(\lvert z_1 \rvert = 0 \text{ or } \infty, \lvert z_2 \rvert = 0 \text{ or } \infty)$.
Property 4:	Stable sequence $\longleftrightarrow$ ROC includes the unit surface, $(\lvert z_1 \rvert = 1, \lvert z_2 \rvert = 1)$.

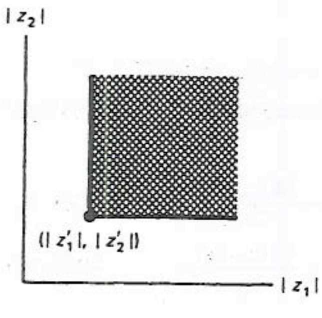


Figure 7.22 Constraint map for the ROC of first-quadrant sequences. For any (z_1', z_2') in the ROC, all (z_1, z_2) in the shaded region is also in the ROC.

that satisfy the constraint map in Fig. 7.22 are those shown in Figs. 7.18 and 7.21. They are the ROCs of $x(n_1, n_2) = a^{n_1} \cdot b^{n_2} \cdot u(n_1, n_2)$ and $x(n_1, n_2) = a^{n_1} \cdot \delta(n_1 - n_2) \cdot u(n_1, n_2)$, both of which are first-quadrant sequences. Constraint maps can also be obtained for other quadrant sequences.

7.3.4 Properties of the z-Transform

Many properties of the z-transform can be obtained from the z-transform definition of Eq. (7.32). Some important properties are listed in Table 7.3. All the properties, except properties 3 and 7, can be viewed as straightforward extensions of the 1-D case. Property 3 applies to separable sequences, and property 7 can be used in determining the z-transform of a first-quadrant sequence obtained by linearly mapping the variables of a wedge sequence.

7.3.5 Inverse z-Transform

The z-transform definition in Eq. (7.32) can be used to determine the z-transform and ROC of a 2-D sequence. As in the 1-D case, using Eq. (7.32) and Cauchy's integral theorem, we can determine the inverse z-transform relation that expresses $x(n_1, n_2)$ as a function of $X(z_1, z_2)$ and its ROC. The inverse z-transform relation is given by

$$x(n_1, n_2) = \frac{1}{(2\pi j)^2} \cdot \oint_{C_1} \oint_{C_2} X(z_1, z_2) \cdot z_1^{n_1 - 1} \cdot z_2^{n_2 - 1} \cdot dz_1 \cdot dz_2 \qquad (7.36)$$

TABLE 7.3 PROPERTIES OF THE z-TRANSFORM

$x(n_1, n_2) \longleftrightarrow X(z_1, z_2)$, ROC: R_x	$y(n_1, n_2) \longleftrightarrow Y(z_1, z_2)$, ROC: R_y
Property 1: Linearity	$a \cdot x(n_1, n_2) + b \cdot y(n_1, n_2) \longleftrightarrow a \cdot X(z_1, z_2) + b \cdot Y(z_1, z_2)$ ROC: at least $R_x \cap R_y$
Property 2: Convolution	$x(n_1, n_2) * y(n_1, n_2) \longleftrightarrow X(z_1, z_2) \cdot Y(z_1, z_2)$ ROC: at least $R_x \cap R_y$
Property 3: Separable sequence	$x_1(n_1) \cdot x_2(n_2) \longleftrightarrow X_1(z_1) \cdot X_2(z_2)$ ROC: $\lvert z_1 \rvert \in$ ROC of $X_1(z_1)$ and $\lvert z_2 \rvert \in$ ROC of $X_2(z_2)$
Property 4: Shift of a sequence	$x(n_1 - m_1, n_2 - m_2) \longleftrightarrow X(z_1, z_2) \cdot z_1^{-m_1} \cdot z_2^{-m_2}$ ROC: R_x with possible exception of $\lvert z_1 \rvert = 0, \infty, \lvert z_2 \rvert = 0, \infty$
Property 5: Differentiation	(a) $-n_1 \cdot x(n_1, n_2) \longleftrightarrow \frac{\partial X(z_1, z_2)}{\partial z_1} z_1$, ROC: R_x (b) $-n_2 \cdot x(n_1, n_2) \longleftrightarrow \frac{\partial X(z_1, z_2)}{\partial z_2} z_2$, ROC: R_x
Property 6: Symmetry properties	(a) $x^*(n_1, n_2) \longleftrightarrow X^*(z_1^*, z_2^*)$, ROC: R_x (b) $x(-n_1, -n_2) \longleftrightarrow X(z_1^{-1}, z_2^{-1})$, ROC: $\lvert z_1^{-1} \rvert, \lvert z_2^{-1} \rvert$ in R_x
Property 7: Linear mapping of variables	$x(n_1, n_2) = y(m_1, m_2)\rvert_{(m_1 = I \cdot n_1 + J \cdot n_2,\, m_2 = K \cdot n_1 + L \cdot n_2)} \longleftrightarrow Y(z_1, z_2) = X(z_1^I \cdot z_2^K, z_1^J \cdot z_2^L)$ ROC: $(\lvert z_1^I \cdot z_2^K \rvert, \lvert z_1^J \cdot z_2^L \rvert)$ in R_x *Note:* For those points of $y(m_1, m_2)$ that do not correspond to any $x(n_1, n_2)$, $y(m_1, m_2)$ is taken to be zero.

where C_1 and C_2 are both in the ROC of $X(z_1, z_2)$, C_1 is a closed contour that encircles in the counterclockwise direction the origin in the z_1-plane for any fixed z_2 on C_2, and C_2 is a closed contour that encircles in the counterclockwise direction the origin in the z_2-plane for any fixed z_1 on C_1.

The conditions that the contours C_1 and C_2 in Eq. (7.36) must satisfy appear to be quite complex, but there is a simple way to determine the contours C_1 and C_2 given the ROC. Suppose $(\lvert z_1' \rvert, \lvert z_2' \rvert)$ lies in the ROC. One set of contours C_1 and C_2 that satisfies the conditions in Eq. (7.36) is

$$\begin{aligned} C_1&: \quad z_1 = \lvert z_1' \rvert \cdot e^{j\omega_1}, \qquad \omega_1\text{: } 0 \text{ to } 2\pi \\ C_2&: \quad z_2 = \lvert z_2' \rvert \cdot e^{j\omega_2}, \qquad \omega_2\text{: } 0 \text{ to } 2\pi \end{aligned} \tag{7.37}$$

If the sequence is stable, so that the ROC includes the unit surface, one possible set of contours is

$$\begin{aligned} C_1:&\quad z_1 = e^{j\omega_1}, \qquad \omega_1: \ 0 \text{ to } 2\pi \\ C_2:&\quad z_2 = e^{j\omega_2}, \qquad \omega_2: \ 0 \text{ to } 2\pi \end{aligned} \tag{7.38}$$

If this set of contours is chosen in Eq. (7.36), then Eq. (7.36) reduces to the inverse Fourier transform relation in Eq. (7.29). This again shows that the Fourier transform representation is a special case of the z-transform representation.

From Eq. (7.36), the result of the contour integration differs depending on the contours. Therefore, a sequence $x(n_1, n_2)$ is not uniquely specified by $X(z_1, z_2)$ alone. Since the contours C_1 and C_2 can be determined from the ROC of $X(z_1, z_2)$, a sequence $x(n_1, n_2)$ is uniquely specified by both $X(z_1, z_2)$ and its ROC:

$$x(n_1, n_2) \quad \longleftrightarrow \quad X(z_1, z_2), \quad \text{ROC} \tag{7.39}$$

In theory, Eq. (7.36) can be used directly in determining $x(n_1, n_2)$ from $X(z_1, z_2)$ and its ROC. In practice, however, it is extremely tedious to perform the contour integration even for simple problems.

For 1-D signals, the approaches that have been used in performing the inverse z-transform operation without explicit evaluation of a contour integral are series expansion, partial fraction expansion, and inverse z-transform using z-transform properties. Among these approaches, the partial fraction expansion method is the only one that can always be used in performing the z-transform operation for any rational z-transform. In this method, the z-transform $X(z)$ is first expressed as a sum of simpler z-transforms by factoring the denominator polynomial as a product of lower-order polynomials. The inverse z-transform is then performed for each of the simpler z-transforms, and the results are added to obtain $x(n)$.

For 2-D signals, unfortunately, the partial fraction expansion method is not a general procedure that can be used to perform the inverse z-transform operation for a rational z-transform. In the 1-D case, the factorization of any 1-D polynomial as a product of lower-order polynomials is guaranteed by the fundamental theorem of algebra. A 2-D polynomial, however, cannot in general be factored as a product of lower-order polynomials. Therefore, a procedure analogous to the 1-D case cannot generally be used. Partly due to the difficulty involved in the partial fraction expansion method, no known practical method exists that can be used to perform the inverse z-transform of a general 2-D rational z-transform.

7.4 DIFFERENCE EQUATION

7.4.1 Linear Constant-Coefficient Difference Equation

Difference equations play a more important role for discrete space systems than differential equations do for analog systems. In addition to representing a wide class of discrete space systems, difference equations can be used to recursively generate their solutions. This use can be exploited in realizing digital filters with infinite-extent unit sample responses.

In this section we consider the class of linear constant coefficient difference equation (LCCDE) of the following form:

$$\sum\sum_{(k_1,k_2)\in R_A} a(k_1,k_2)\cdot y(n_1-k_1,n_2-k_2) = \sum\sum_{(k_1,k_2)\in R_B} b(k_1,k_2)\cdot x(n_1-k_1,n_2-k_2) \tag{7.40}$$

where $a(k_1,k_2)$ and $b(k_1,k_2)$ are known sequences, R_A represents the region in (k_1,k_2) such that $a(k_1,k_2)$ is nonzero, and R_B is similarly defined.

The LCCDE alone does not specify a system since there are many solutions of $y(n_1,n_2)$ in Eq. (7.40) for a given $x(n_1,n_2)$. For example, if $y_1(n_1,n_2)$ is a solution to $y(n_1,n_2)=\frac{1}{2}y(n_1-1,n_2+1)+\frac{1}{2}y(n_1+1,n_2-1)+x(n_1,n_2)$, then so is $y_1(n_1,n_2)+f(n_1+n_2)$ for any function f. To uniquely specify a solution, we need a set of boundary conditions. Since the boundary conditions have to determine specific functional forms as well as the constants associated with the functions, they are typically an infinite number of points in the output $y(n_1,n_2)$. This aspect differs fundamentally from the 1-D case. In the 1-D case, the LCCDE specifies a solution within arbitrary constants. For example, an Nth-order 1-D LCCDE specifies a solution within N constants, and therefore N initial conditions (N points in the output $y(n)$) are generally sufficient to uniquely specify a solution. In the 2-D case, we need a set of boundary conditions that are typically an infinite number of points in the output $y(n_1,n_2)$.

7.4.2 Difference Equation with Boundary Conditions

The problem of solving an LCCDE with boundary conditions can be stated as follows: Given $x(n_1,n_2)$, and $y(n_1,n_2)$ for $(n_1,n_2)\in R_{BC}$, find the solution to

$$\sum\sum_{(k_1,k_2)\in R_A} a(k_1,k_2)\cdot y(n_1-k_1,n_2-k_2) = \sum\sum_{(k_1,k_2)\in R_B} b(k_1,k_2)\cdot x(n_1-k_1,n_2-k_2) \tag{7.41}$$

One approach that can be used in solving a 1-D LCCDE with initial conditions is to obtain a homogeneous solution and a particular solution, determine the total solution as a sum of these, and then impose initial conditions. For the 2-D case, unfortunately, this approach cannot be used. First, there is no general procedure to obtain the homogeneous solution. The homogeneous solution consists of unknown functions, and the specific functional form of $k\cdot a^n$, used in the 1-D case, cannot be used for the 2-D case. Second, the particular solution cannot generally be obtained by inspection or by the z-transform method since there is no practical procedure for performing the inverse z-transform operation for the 2-D case. Furthermore, determining the unknown functions in the homogeneous solution by imposing the boundary conditions (an infinite number of known values of $y(n_1,n_2)$) is not a simple linear problem.

Another approach in solving Eq. (7.41) is to compute $y(n_1,n_2)$ recursively, which is how it is typically done on a computer. To illustrate this approach, consider the following 2-D LCCDE with boundary conditions (BC):

$$y(n_1, n_2) = y(n_1 - 1, n_2) + y(n_1, n_2 - 1) + y(n_1 - 1, n_2 - 1) + x(n_1, n_2)$$
$$x(n_1, n_2) = \delta(n_1, n_2) \tag{7.42}$$
$$\text{BC: } y(n_1, n_2) = 1, \qquad \text{for } n_1 < 0 \text{ or } n_2 < 0$$

The output $y(n_1, n_2)$ can be obtained recursively as follows:

$$\begin{aligned} y(0,0) &= y(-1,0) + y(0,-1) + y(-1,-1) + x(0,0) = 4 \\ y(1,0) &= y(0,0) + y(1,-1) + y(0,-1) + x(1,0) = 6 \\ y(0,1) &= y(-1,1) + y(0,0) + y(-1,0) + x(0,1) = 6 \\ y(2,0) &= \cdots \\ &\vdots \end{aligned} \tag{7.43}$$

Even though the approach to computing the output recursively appears to be quite simple, the proper choice of the boundary conditions so that the LCCDE with boundary conditions has a unique solution is not straightforward. In the 1-D case, N initial conditions are typically both necessary and sufficient for an Nth-order difference equation. In the 2-D case, the choice of the boundary conditions so that the LCCDE will have a unique solution is not too obvious.

Suppose we have chosen the boundary conditions such that the LCCDE has a unique solution and therefore we can consider the LCCDE with boundary conditions as a system. In both the 1-D and 2-D cases, the system is in general neither linear nor shift-invariant. The difference equation is of interest to us primarily because it is the only practical way to realize an infinite impulse response (IIR) digital filter. Since a digital filter is a linear shift-invariant system, we need to force the difference equation to become a linear shift-invariant system. We can do this by choosing a proper set of boundary conditions. In the 1-D case, one way to force a difference equation to be a linear shift-invariant system is to impose an initial condition known as an initial rest condition. An initial rest condition is defined to be an initial condition obtained by requiring the output $y(n)$ to be zero for $n < n_0$ whenever the input $x(n)$ is zero for $n < n_0$, and it also forces the resulting system to be causal. Even though the extension is not quite straightforward, it is possible to extend the general idea behind the initial rest condition to the 2-D case. This is discussed in the following section.

7.4.3 Difference Equation as a Linear Shift-Invariant System

One way to force an LCCDE with boundary conditions to be a linear shift-invariant system so that it can be used in realizing an IIR filter is to choose the boundary conditions using the following three steps.

1. Interpret the LCCDE as a specific computational procedure.
2. Determine R_{BC}, the region (n_1, n_2) for which the boundary conditions are applied, as follows:
   (a) Determine R_h, the region of support of the unit sample response $h(n_1, n_2)$.

(b) Determine R_y, the region of support for the output $y(n_1, n_2)$, from R_h and R_x, where R_x is similarly defined.
(c) R_{BC}: all $(n_1, n_2) \notin R_y$.

3. Boundary conditions: Set $y(n_1, n_2) = 0$ for $(n_1, n_2) \in R_{BC}$.

Step 1. In this step, we interpret a difference equation as a specific computational procedure. The best way to explain this step is by looking at a specific example. Consider the following LCCDE:

$$y(n_1, n_2) + 2 \cdot y(n_1 - 1, n_2) + 3y(n_1, n_2 - 1) + 4(n_1 - 1, n_2 - 1) = x(n_1, n_2) \tag{7.44}$$

Since there are four terms of the form $y(n_1 - k_1, n_2 - k_2)$, we obtain four equations by leaving only one of the four terms on the left-hand side of the equation, as follows:

$$y(n_1, n_2) = -2y(n_1 - 1, n_2) - 3y(n_1, n_2 - 1) - 4 \cdot y(n_1 - 1, n_2 - 1) + x(n_1, n_2) \tag{7.45a}$$

$$y(n_1 - 1, n_2) = -\tfrac{1}{2}y(n_1, n_2) - \tfrac{3}{2}y(n_1, n_2 - 1) - 2y(n_1 - 1, n_2 - 1) + \tfrac{1}{2}x(n_1, n_2) \tag{7.45b}$$

$$y(n_1, n_2 - 1) = -\tfrac{1}{3}y(n_1, n_2) - \tfrac{2}{3}y(n_1 - 1, n_2) - \tfrac{4}{3}y(n_1 - 1, n_2 - 1) + \tfrac{1}{3}x(n_1, n_2) \tag{7.45c}$$

$$y(n_1 - 1, n_2 - 1) = -\tfrac{1}{4}y(n_1, n_2) - \tfrac{1}{2}y(n_1 - 1, n_2) - \tfrac{3}{4}y(n_1, n_2 - 1) + \tfrac{1}{4}(n_1, n_2) \tag{7.45d}$$

By a simple change of variables, Eqs. (7.45) can be rewritten so that the left-hand side of each equation has the form $y(n_1, n_2)$:

$$y(n_1, n_2) = -2y(n_1 - 1, n_2) - 3(n_1, n_2 - 1) - 4y(n_1 - 1, n_2 - 1) + x(n_1, n_2) \tag{7.46a}$$

$$\begin{aligned} y(n_1, n_2) = &-\tfrac{1}{2}y(n_1 + 1, n_2) - \tfrac{3}{2}y(n_1 + 1, n_2 - 1) - 2y(n_1, n_2 - 1) \\ &+ \tfrac{1}{2}x(n_1 + 1, n_2) \end{aligned} \tag{7.46b}$$

$$\begin{aligned} y(n_1, n_2) = &-\tfrac{1}{3}y(n_1, n_2 + 1) - \tfrac{2}{3}y(n_1 - 1, n_2 + 1) - \tfrac{4}{3}y(n_1 - 1, n_2) \\ &+ \tfrac{1}{3}(n_1, n_2 + 1) \end{aligned} \tag{7.46c}$$

$$\begin{aligned} y(n_1, n_2) = &-\tfrac{1}{4}y(n_1 + 1, n_2 + 1) - \tfrac{1}{2}y(n_1, n_2 + 1) - \tfrac{3}{4}y(n_1 + 1, n_2) \\ &+ \tfrac{1}{4}x(n_1 + 1, n_2 + 1) \end{aligned} \tag{7.46d}$$

Even though all four equations in Eq. (7.46) are the same LCCDEs, they correspond to four different specific computational procedures by proper interpretation. The interpretation we use is that the left-hand side $y(n_1, n_2)$ is always computed from the right-hand side expression for all (n_1, n_2). When this interpretation is strictly followed, then each of the four equations in Eq. (7.46) corresponds to a different computational procedure. This will become clearer when we discuss step 2.

It is often convenient to represent a specific computational procedure pictorially. Specifically, to pictorially represent the specific computational procedure corresponding to Eq. (7.46a), we consider computing $y(0,0)$. Since $y(n_1, n_2)$ on the left-hand side is always computed from the right-hand side, $y(0,0)$ in this case is computed as

$$y(0,0) \longleftarrow -2y(-1,0) - 3y(0,-1) - 4y(-1,-1) + x(0,0) \qquad (7.47)$$

We have used $\leftarrow$ to emphasize that $y(0,0)$ is always computed from the right-hand side. Equation (7.47) is represented in Fig. 7.23. The value $y(0,0)$ that is computed is denoted by $\times$ in Fig. 7.23(a). The values $y(-1,0)$, $y(0,-1)$, and $y(-1,-1)$ that are used in obtaining $y(0,0)$ are marked as solid dots (•) in the figure, with the proper coefficient attached to the corresponding point. The value $x(0,0)$ used in obtaining $y(0,0)$ is marked as a solid dot in Fig. 7.23(b). To compute $y(0,0)$, therefore, we look at $y(n_1, n_2)$ and $x(n_1, n_2)$ at the points marked by solid dots, multiply each value by the corresponding scaling factor indicated, and sum all the terms. Figure 7.23(a) is called an *output mask* and Fig. 7.23(b) is called an *input mask* since they are "masks" that are applied to the output and input to compute $y(0,0)$. We note that the output mask always has $\times$ at the origin, but the input mask does not have $\times$ anywhere.

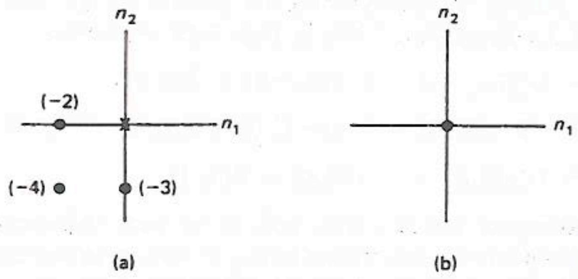


Figure 7.23 (a) Output mask and (b) input mask corresponding to Eq. (7.47). Note that the output mask always has an $\times$ at the origin, but the input mask does not have an $\times$ anywhere.

Even though the output and input masks are sketched for the case when $y(0,0)$ is computed, they are also very useful in visualizing what happens when other points of $y(n_1, n_2)$ are computed. Suppose we wish to compute $y(5,3)$ using the same computational procedure as in Fig. 7.23. The points that are used in determining $y(5,3)$ are indicated in Fig. 7.24, with the proper scaling factors attached. Figure 7.24 is simply a shifted version of Fig. 7.23.

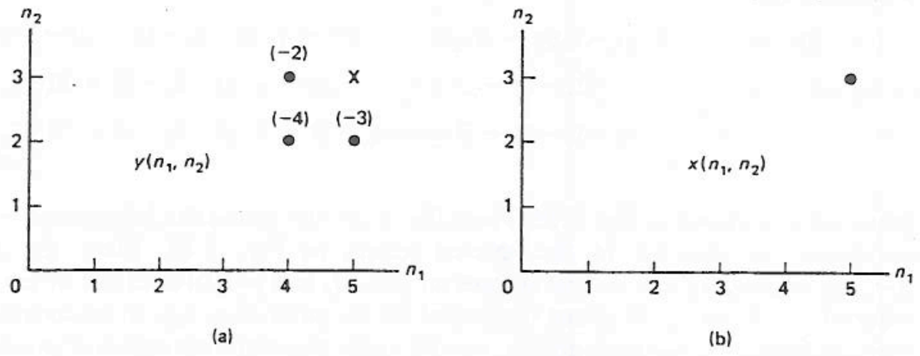


Figure 7.24 (a) Output mask and (b) input mask in Fig. 7.23 shifted to illustrate the computation of $y(5,3)$.

From the above discussion, an LCCDE can lead to many different specific computational procedures. Which procedure is chosen from these many possibilities is determined from the specific application context. We discuss this point in the specific example following steps 2 and 3.

Step 2. In this step, we determine R_y, the region of support of the output $y(n_1, n_2)$. To determine R_y, we first determine R_h, the region of support of the unit sample response. To illustrate how R_h is determined, consider the following computational procedure:

$$\begin{aligned} y(n_1, n_2) \longleftarrow & -2y(n_1 - 1, n_2) - 3y(n_1, n_2 - 1) \\ & - 4y(n_1 - 1, n_2 - 1) + x(n_1, n_2) \end{aligned} \tag{7.48}$$

The output and input masks for this computational procedure were shown in Fig. 7.23. The region of support R_h is the region of (n_1, n_2) for which $y(n_1, n_2)$ is influenced by the pulse $\delta(n_1, n_2)$ when we set $x(n_1, n_2) = \delta(n_1, n_2)$. Consider $y(0, 0)$. From Eq. (7.48) or Fig. 7.23, we see that $y(0, 0)$ is influenced by $y(-1, 0)$, $y(0, -1)$, $y(-1, -1)$, and $\delta(0, 0)$. Clearly $y(0, 0)$ is influenced by the pulse $\delta(n_1, n_2)$. Now consider $y(1, 0)$, $y(0, 1)$, and $y(1, 1)$. From Eq. (7.48) or Fig. 7.23, we have

$$\begin{aligned} y(1,0) &\longleftarrow -2\cdot y(0,0) - 3\cdot y(1,-1) - 4\cdot y(0,-1) + \delta(1,0) \\ y(0,1) &\longleftarrow -2\cdot y(-1,1) - 3\cdot y(0,0) - 4\cdot y(-1,0) + \delta(0,1) \\ y(1,1) &\longleftarrow -2\cdot y(0,1) - 3\cdot y(1,0) - 4\cdot y(0,0) + \delta(1,1) \end{aligned} \tag{7.49}$$

Since $\delta(n_1, n_2)$ has already influenced $y(0, 0)$, and $y(0, 0)$ in turn influences $y(1, 0), y(0, 1)$, and $y(1, 1)$, the above three output values in Eq. (7.49) are influenced by the pulse $\delta(n_1, n_2)$. Now consider $y(-1, 0)$. From Eq. (7.48) or Fig. 7.23, we see that $y(-1, 0)$ is obtained from

$$y(-1,0) \longleftarrow -2\cdot y(-2,0) - 3\cdot y(-1,-1) - 4\cdot y(-2,-1) + \delta(-1,0) \tag{7.50}$$

This is shown in Fig. 7.25. The terms that influence $y(-1, 0)$ in Eq. (7.50) are obtained from

$$\begin{aligned} y(-2,0) &\longleftarrow -2\cdot y(-3,0) - 3\cdot y(-2,-1) - 4\cdot y(-3,-1) + \delta(-2,0) \\ y(-1,-1) &\longleftarrow -2\cdot y(-2,-1) - 3\cdot y(-1,-2) - 4\cdot y(-2,-2) + \delta(-1,-1) \\ y(-2,-1) &\longleftarrow -2\cdot y(-3,-1) - 3\cdot y(-2,-2) - 4\cdot y(-3,-2) + \delta(-2,-1) \end{aligned} \tag{7.51}$$

These are also shown in Fig. 7.25. From Fig. 7.25, the points that influence $y(-1, 0)$ are shown as $y(n_1, n_2)$ in the shaded region in Fig. 7.26. Since the pulse $x(n_1, n_2) = \delta(n_1, n_2)$ has its first impact on $y(0, 0)$, and $y(0, 0)$ does not in any way affect $y(-1, 0)$, $y(-1, 0)$ is not influenced by the pulse $\delta(n_1, n_2)$. If we consider all other points of $y(n_1, n_2)$ analogously, we can easily argue that the region of (n_1, n_2) for which $y(n_1, n_2)$ is influenced by $x(n_1, n_2) = \delta(n_1, n_2)$ is only the first-quadrant region. This is the region R_h. In essence, the pulse $\delta(n_1, n_2)$ has a direct effect only on $y(0, 0)$.

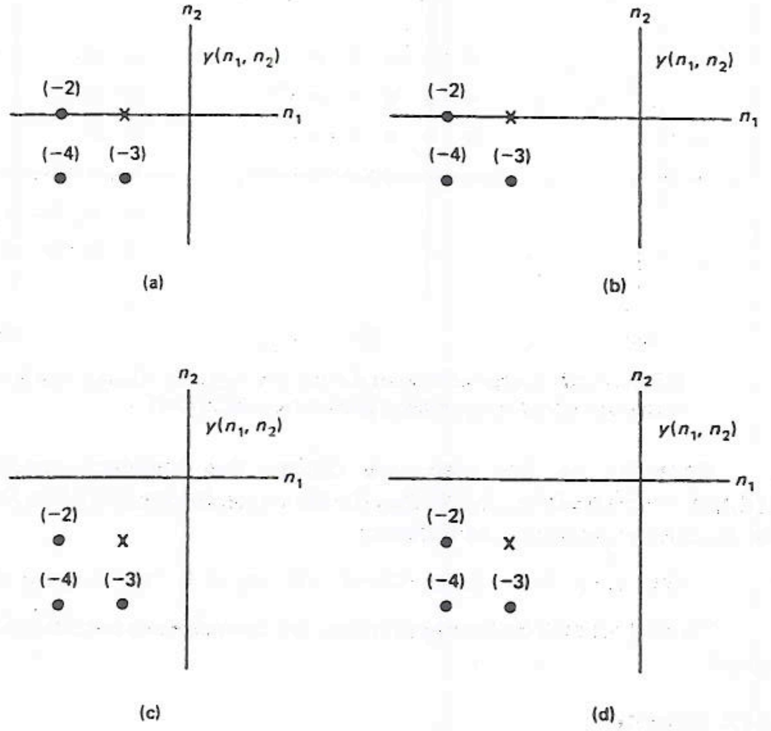


Figure 7.25 Computation of $y(-1,0)$ and its neighborhood values: (a) $y(-1,0)$; (b) $y(-2,0)$; (c) $y(-1,-1)$; (d) $y(-2,-1)$.

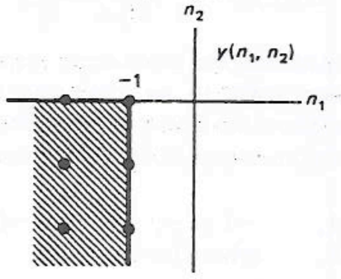


Figure 7.26 The region that influences $y(-1,0)$.

Because of the specific computational procedure in Eq. (7.48) and Fig. 7.23, $y(0,0)$ influences only the first-quadrant region.

Once R_h is determined, R_y can be obtained from R_h and R_x. Suppose $x(n_1, n_2)$ is the sequence shown in Fig. 7.27(a). By convolving $x(n_1, n_2)$ and $h(n_1, n_2)$, R_y can be easily determined. For R_h considered above, R_y is given by the shaded region in Fig. 7.27(b). R_{BC}, the region (n_1, n_2) for which the boundary conditions are applied, is determined to be all (n_1, n_2) outside R_y. In the current example, R_{BC} is given by the shaded region in Fig. 7.27(c).

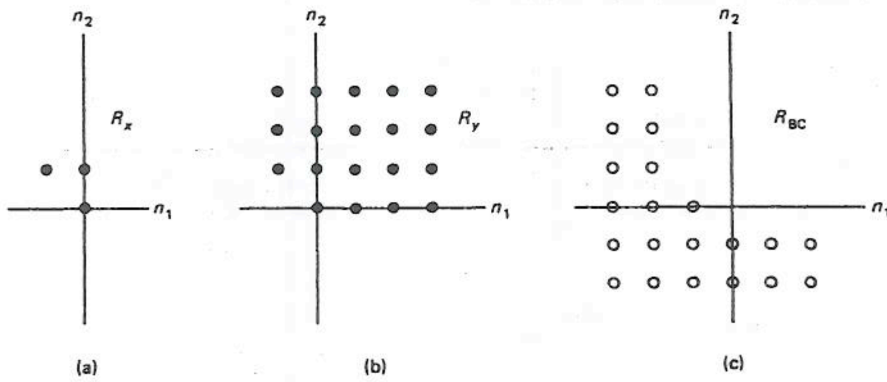


Figure 7.27 Region of support for (a) $x(n_1, n_2)$, (b) $y(n_1, n_2)$ and (c) boundary conditions for the computational procedure in Eq. (7.48).

Step 3. In this step, we choose the boundary conditions such that $y(n_1, n_2) = 0$ for all $(n_1, n_2) \in R_{BC}$. In the example considered in Step 2, we choose the boundary conditions as follows:

$$y(n_1, n_2) = 0 \qquad \text{for } n_1 < -1 \quad \text{or} \quad n_2 < 0 \quad \text{or} \quad (n_1, n_2) = (-1, 0)$$

To illustrate the three steps further, we consider one specific example in the next section.

7.4.4 Example

Suppose we have designed an IIR digital filter whose system function $H(z_1, z_2)$ is given by

$$H(z_1, z_2) = \frac{1}{1 + \frac{1}{2} z_1^{-1} \cdot z_2^{-1}} \tag{7.52}$$

The IIR filter was designed by making the unit sample response of the designed system as close as possible in some sense to an ideal unit sample response that is a first-quadrant sequence. Therefore, we know that the filter is a first-quadrant support system. We wish to determine the output of the filter when the input $x(n_1, n_2)$ is given by

$$x(n_1, n_2) = \begin{cases} 1, & -1 \le n_1 \le 1 \text{ and } -1 \le n_2 \le 1 \\ 0, & \text{otherwise} \end{cases} \tag{7.53}$$

as shown in Fig. 7.28.

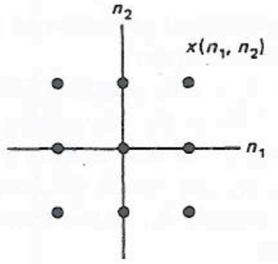


Figure 7.28 An input sequence $x(n_1, n_2)$ to the system of Eq. (7.52).

Since the only practical way to implement a general IIR filter is by using a difference equation, we first convert Eq. (7.52) to a difference equation as follows:

$$H(z_1, z_2) = \frac{Y(z_1, z_2)}{X(z_1, z_2)} = \frac{1}{1 + \frac{1}{2}z_1^{-1} \cdot z_2^{-1}}$$

$$Y(z_1, z_2) + \tfrac{1}{2} \cdot Y(z_1, z_2) \cdot z_1^{-1} \cdot z_2^{-1} = X(z_1, z_2) \tag{7.54}$$

$$y(n_1, n_2) + \tfrac{1}{2} \cdot y(n_1 - 1, n_2 - 1) = x(n_1, n_2)$$

Since the IIR filter is a linear shift-invariant system, we choose the proper set of boundary conditions so that the LCCDE becomes a linear shift-invariant system. Two specific computational procedures correspond to the LCCDE in Eq. (7.54):

$$y(n_1, n_2) \longleftarrow -\tfrac{1}{2}y(n_1 - 1, n_2 - 1) + x(n_1, n_2) \tag{7.55}$$

$$y(n_1, n_2) \longleftarrow -2 \cdot y(n_1 + 1, n_2 + 1) + 2 \cdot x(n_1 + 1, n_2 + 1) \tag{7.56}$$

The output and input masks corresponding to each of the two computational procedures in Eqs. (7.55) and (7.56) are shown in Fig. 7.29. The region of support of $h(n_1, n_2)$ for each of the two computational procedures is shown in Fig. 7.30. Since we know that the filter is a first-quadrant support system, we choose the computational procedure given by Eq. (7.55). To determine the boundary conditions for the computational procedure chosen, we determine R_y, the region of support for the output sequence $y(n_1, n_2)$, from R_x in Fig. 7.28 and R_h in Fig. 7.30(a). The region R_y is shown in Fig. 7.31(a). The boundary conditions that we use are shown in Fig. 7.31(b). With the boundary conditions shown in Fig. 7.31(b), the output $y(n_1, n_2)$ can be recursively computed from Eqs. (7.53) and (7.55). The result is shown in Fig. 7.32. If we double the input, the output will also double. If we shift the input, then the output will be shifted by a corresponding amount. This is consistent with the fact that the computational procedure is a linear shift-invariant system.

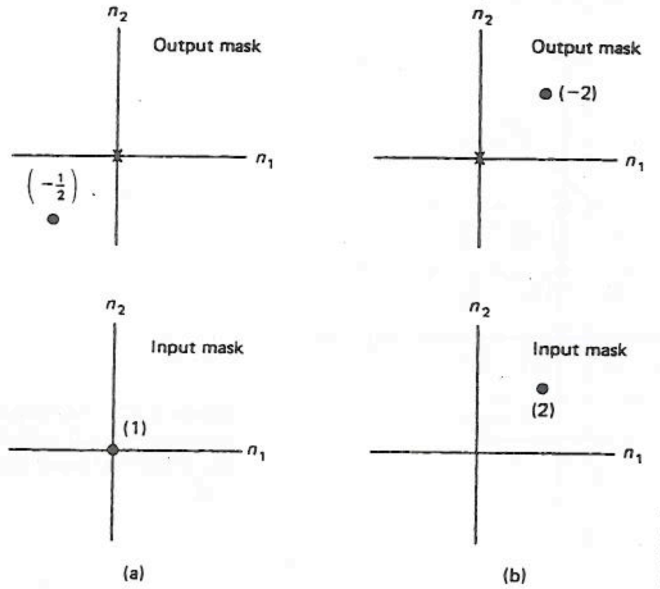


Figure 7.29 Output and input masks for the computational procedure given by (a) Eq. (7.55) and (b) Eq. (7.56).

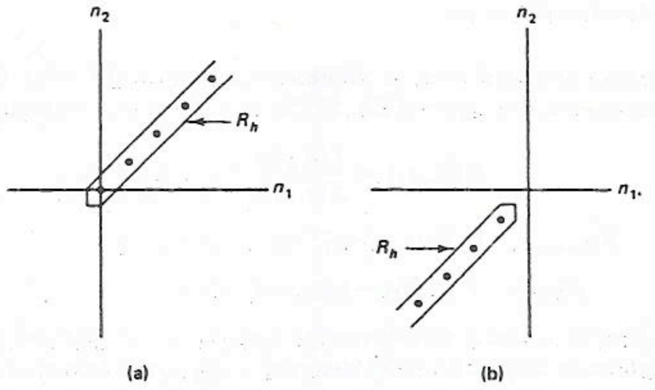


Figure 7.30 The regions of support for $h(n_1, n_2)$ corresponding to the two computational procedures given by (a) Eq. (7.55) and (b) Eq. (7.56).

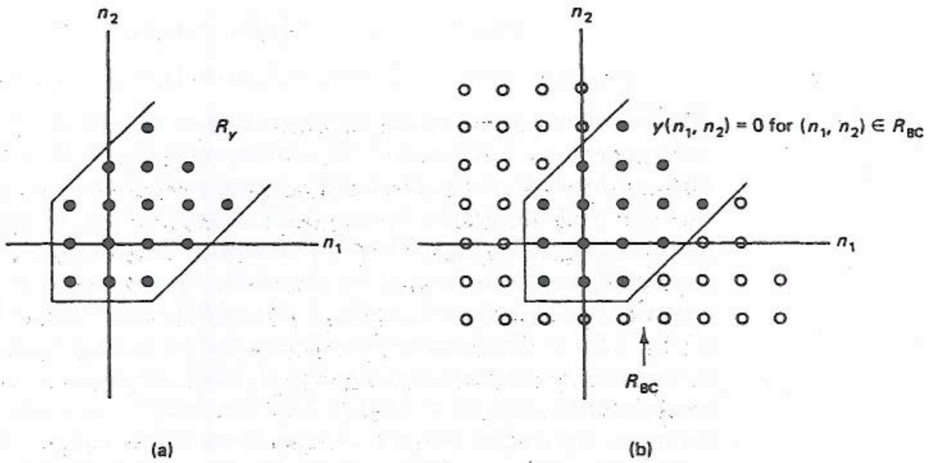


Figure 7.31 The regions of support for (a) $y(n_1, n_2)$ and (b) boundary conditions, for the computational procedure given by Eq. (7.55) and the input $x(n_1, n_2)$ in Fig. 7.28.

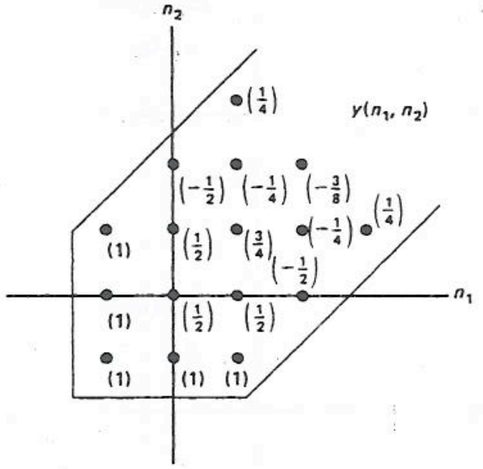


Figure 7.32 The output $y(n_1, n_2)$ for the computational procedure given by Eq. (7.55) and input $x(n_1, n_2)$ in Fig. 7.28.

7.4.5 Recursive Computability

The LCCDE with boundary conditions plays a particularly important role in digital signal processing since it is the only practical way to realize an IIR filter. As we discussed in the previous sections, the LCCDE can be used as a recursive procedure in computing the output. We define a system to be recursively computable when there exists a path we can follow in computing every output point recursively, one at a time. The example in Section 7.4.4 corresponds to a recursively computable system.

From the definition of a recursively computable system, we can easily show that not all computational procedures resulting from LCCDEs are recursively computable. For example, consider a computational procedure whose output mask is shown in Fig. 7.33. From the output mask, it is clear that computing $y(0, 0)$ requires $y(1, 0)$ and computing $y(1, 0)$ requires $y(0, 0)$. Therefore, we cannot compute $y(0, 0)$ and $y(1, 0)$ one at a time recursively, and the computational procedure in Fig. 7.33 is not a recursively computable system. For a finite-extent input $x(n_1, n_2)$, we can show that a system is recursively computable if the output mask has a wedge support. Examples of wedge support output masks are shown in Fig. 7.34. The types of output masks shown in Figs. 7.34(b) and (c) are called "nonsymmetric half-plane" output masks. Examples of output masks that do not have wedge support are shown in Fig. 7.35.

For a recursively computable system, we can follow many different paths in computing all the output values we need. Even though $y(n_1, n_2)$ can be computed in many different orders, the result does not depend on the specific order that is used.

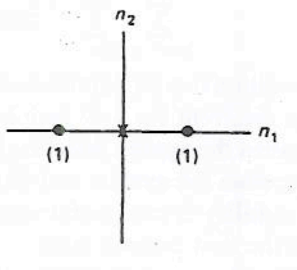


Figure 7.33 An example of an output mask of a system that is not recursively computable.

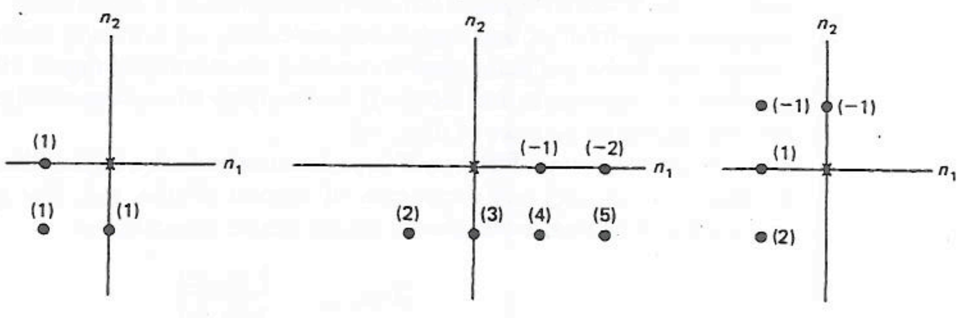


Figure 7.34 Examples of output masks of recursively computable systems.

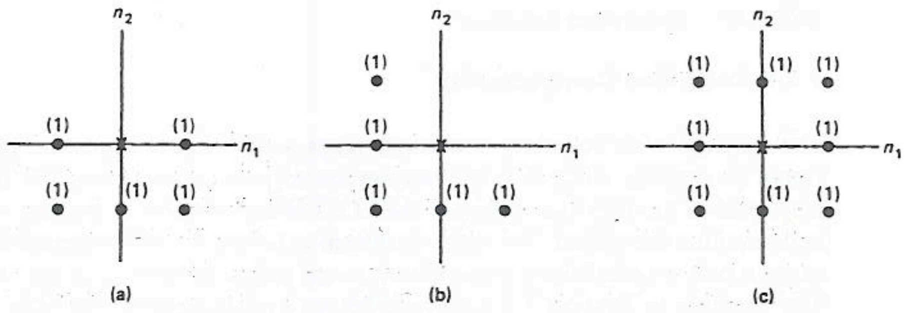


Figure 7.35 Examples of output masks of systems that are not recursively computable.

7.5 STABILITY

7.5.1 Stability Problem

When a discrete space system such as a digital filter is designed, an important consideration is the stability of the system. In this section, we consider the problem of testing the stability of a discrete space system.

As discussed in Section 7.1.4, a system is considered stable in the bounded input–bounded output (BIBO) sense if and only if a bounded input always leads to a bounded output. For a linear shift-invariant system, a necessary and sufficient condition for the system to be stable is that its unit sample response $h(n_1, n_2)$ be absolutely summable:

$$\sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} |h(n_1, n_2)| < \infty$$

Even though this condition is a straightforward extension of 1-D results, issues related to stability are quite different for 1-D and 2-D results.

Often we are given the system function $H(z_1, z_2)$ and information about its ROC, and we wish to determine the system stability. If the ROC is explicitly given, determining the system stability is straightforward because a system is stable if and only if the ROC includes the unit surface ($|z_1| = |z_2| = 1$). Unfortunately, however, the ROC is seldom explicitly given; typically, only implicit information about the ROC is available. When the system function corresponds to a digital filter, for example, the region of support of its unit sample response $h(n_1, n_2)$ is usually known from the filter design step. Since our main interest is testing the stability of digital filters, the stability problem we consider in this section is to determine the system stability given $H(z_1, z_2)$ and the region of support of $h(n_1, n_2)$.

When the system function $H(z_1, z_2)$ corresponds to a digital filter, restrictions are imposed on $H(z_1, z_2)$ and the region of support of $h(n_1, n_2)$. One restriction is that $H(z_1, z_2)$ is a rational z-transform, which can be expressed as

$$H(z_1, z_2) = \frac{B(z_1, z_2)}{A(z_1, z_2)} \tag{7.57}$$

where $A(z_1, z_2)$ and $B(z_1, z_2)$ are finite-order polynomials in z_1 and z_2. Another restriction is that the system is recursively computable. With these two restrictions, the system can be realized by an LCCDE with boundary conditions, and the output can be computed recursively one at a time.

In the 1-D case, when the system function $H(z)$ is expressed as $B(z)/A(z)$ where there are no common factors between $A(z)$ and $B(z)$, $B(z)$ does not affect the system stability. In the 2-D case, however, the presence of $B(z_1, z_2)$ in Eq. (7.57) can stabilize an otherwise unstable system, even when there are no common factors between $A(z_1, z_2)$ and $B(z_1, z_2)$. This case occurs very rarely, there is no known procedure that can be used when such a case does occur, and an unstable system stabilized by $B(z_1, z_2)$ is unstable for all practical purposes. We will therefore assume that the numerator polynomial $B(z_1, z_2)$ does not affect the system stability. To make this explicit, we'll assume that $B(z_1, z_2) = 1$.

When the input $x(n_1, n_2)$ is a finite-extent sequence, which is the case of our primary interest, the recursive computability requires that the output mask has wedge shape support. This in turn requires the unit sample response $h(n_1, n_2)$ to have wedge shape support when $B(z_1, z_2) = 1$. As discussed in Section 7.1.4, it is always possible to find a linear mapping of variables that transforms a wedge sequence to a first-quadrant sequence without affecting the stability of the sequence. Therefore, stability results that apply to first-quadrant sequences can be used for all recursively computable systems. In our approach, we will first transform a recursively computable system to a first-quadrant support system by a linear mapping of variables. This transformation changes the system function $H(z_1, z_2)$ to a new system function $H'(z_1, z_2)$. We will then apply to $H'(z_1, z_2)$ stability results that apply to first-quadrant support systems. As we have discussed, the reason for first transforming a wedge support system to a first-quadrant support system is that developing stability results that apply to first-quadrant support systems is much easier notationally and conceptually than developing stability results that apply to wedge support systems.

Given the preceding discussion, the stability problem we will consider can be stated as follows.

Problem: Given $H(z_1, z_2) = 1/[A(z_1, z_2)]$ and assuming that $h(n_1, n_2)$ is a first-quadrant sequence, determine the system stability.

In the following sections, we discuss several stability theorems and use them to solve the problem of determining the system stability.

7.5.2 Stability Theorems

In the 1-D case, the problem of testing the stability of a causal system whose system function is given by $H(z) = 1/[A(z)]$ is quite straightforward. Since a 1-D polynomial $A(z)$ can always be factored as a product of first-order polynomials, we can always determine the poles of $H(z)$. The stability of the causal system is equivalent to the condition that all the poles are inside the unit circle.

A similar approach cannot be used in testing the stability of a 2-D first-quadrant support system. The approach just described requires the specific locations of all poles. Since a 2-D polynomial $A(z_1, z_2)$ in general cannot be factored as a product of lower-order polynomials, it is extremely difficult to determine all the pole surfaces of

$H(z_1, z_2) = 1/[A(z_1, z_2)]$, and the approach based on explicit determination of all pole surfaces has not led to successful practical procedures of testing the system stability. In this section, we will discuss theorems that can be used in developing practical procedures for testing the stability of a 2-D first-quadrant support system without explicit determination of all pole surfaces.

Shanks' Theorem. Shanks' theorem does not directly lead to practical stability testing procedures, but it is one of the earliest such theorems, is conceptually very simple, and has led to other stability theorems, which we will discuss later. In addition, this theorem shows that a result that is not useful for 1-D signals can be very useful when it is extended to the 2-D case.

In the 1-D case, the stability of a causal system with system function $H(z) = 1/[A(z)]$ is equivalent to requiring all the poles of $H(z)$ to be within the unit circle:

$$\text{Stability} \longleftrightarrow \text{All poles (solutions to } A(z) = 0) \text{ are inside } |z| = 1 \tag{7.58}$$

An equivalent statement to Eq. (7.58) is given by

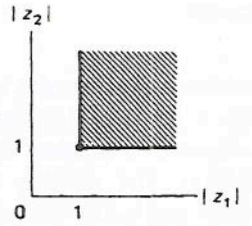
$$\text{Stability} \longleftrightarrow A(z) \neq 0 \quad \text{for } |z| \geq 1 \tag{7.59}$$

If all the poles are inside $|z| = 1$, then $A(z)$ cannot be zero for any $|z| \geq 1$. If $A(z) \neq 0$ for any $|z| \geq 1$, then all the poles must be inside $|z| = 1$. Therefore, Eqs. (7.58) and (7.59) are equivalent conditions.

Even though Eqs. (7.58) and (7.59) are equivalent statements, their implications for testing system stability are quite different. The condition in Eq. (7.58) suggests a procedure where we explicitly determine all pole locations and see if they all are inside $|z| = 1$. The condition in Eq. (7.59), however, suggests a procedure where we evaluate $A(z)$ for each z such that $|z| \geq 1$ and see if $A(z)$ is zero for any $|z| \geq 1$. This requires a search in the 2-D plane. In the 1-D case, the procedure suggested by Eq. (7.58) is extremely simple. Therefore, the procedure suggested by Eq. (7.59), which requires a 2-D search, is not useful. In the 2-D case, however, the procedure suggested by Eq. (7.58) is extremely difficult. The extension of Eq. (7.59) to the 2-D case is Shanks' theorem.

Shanks' theorem can be stated as follows:

$$\text{Stability} \longleftrightarrow A(z_1, z_2) \neq 0 \quad \text{for any } |z_1|, |z_2| \geq 1 \tag{7.60}$$

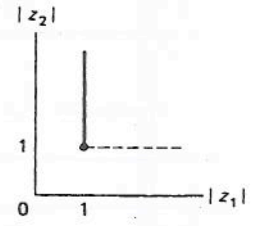


The result in Eq. (7.60) is simple to demonstrate. To show that stability implies the condition in Eq. (7.60), we first note that stability implies that the unit surface $(|z_1| = 1, |z_2| = 1)$ is in the inside of the ROC of $H(z_1, z_2)$. Because $h(n_1, n_2)$ is a first-quadrant sequence, the ROC of $H(z_1, z_2)$ has to satisfy the conditions given by the constraint map in Fig. 7.22. Therefore all (z_1, z_2) such that $|z_1| \geq 1, |z_2| \geq 1$ has to be in the ROC. Since the ROC cannot have any pole surfaces $(A(z_1, z_2) = 0)$, $A(z_1, z_2) \neq 0$ for any $|z_1|, |z_2| \geq 1$. To show that the condition in Eq. (7.60) implies the system stability, we note that this condition implies that there are no pole surfaces for any $|z_1|, |z_2| \geq 1$. Because $h(n_1, n_2)$ is a first-quadrant sequence, the ROC of $H(z_1, z_2)$ has to satisfy the constraint map in Fig. 7.22. This requirement combined with the property that the ROC is bounded by pole surfaces implies that the ROC includes the unit surface, which in turn implies the system stability.

The condition in Eq. (7.60) suggests a procedure where we evaluate $A(z_1, z_2)$ in the 4-D space $(|z_1| \geq 1, |z_2| \geq 1)$ shown in the shaded region in Eq. (7.60). The search in 4-D space is, of course, a tremendous amount of work, and this procedure itself cannot be used in practice. Other theorems, however, state that the space where we search is considerably smaller than the 4-D space in Shanks' theorem. Since the proofs of these theorems are quite involved and the proofs themselves do not provide much insight into the theorems, we state the theorems with our interpretations but without proof.

Huang's Theorem. Huang's theorem can be stated as follows:

$$\text{Stability} \longleftrightarrow \begin{array}{lll} \text{(a)} & A(z_1, z_2) \neq 0 & \text{for } |z_1| = 1, |z_2| \geq 1 \\ \text{(b)} & A(z_1, z_2) \neq 0 & \text{for } |z_1| \geq 1, z_2 = 1 \end{array} \quad (7.61)$$



To satisfy condition (a), we need to ensure that $A(z_1, z_2)$ is not zero for any (z_1, z_2) such that $|z_1| = 1$ and $|z_2| \geq 1$. This requires a 3-D search, where the space to be searched is shown by a solid dark line in the figure in Eq. (7.61). To satisfy condition (b), we need to ensure that $A(z_1, z_2)$ is not zero for any (z_1, z_2) such that $|z_1| \geq 1$ and $z_2 = 1$. This requires a 2-D search, where the space to be searched is shown by the dotted line in the figure. The dotted line is used to emphasize that this is a 2-D search problem, where the search is required in the 2-D subspace of the 3-D space corresponding to $(|z_1| \geq 1, |z_2| = 1)$.

The 3-D search problem corresponding to condition (a) can be substituted for by many 1-D stability tests. To satisfy condition (a), we need to make sure that $A(z_1, z_2) \neq 0$ in the 3-D space corresponding to $(|z_1| = 1, |z_2| \geq 1)$. One approach to satisfy this condition consists of two steps:

1. Solve all (z_1, z_2) such that $A(|z_1| = 1, z_2) = 0$. This is equivalent to solving all (ω_1, z_2) such that $A(e^{j\omega_1}, z_2) = 0$.
2. Check if all $|z_2|$ obtained in step 1 are less than 1.

In step 1, we determine all $(|z_1| = 1, z_2)$ such that $A(|z_1| = 1, z_2) = 0$. The solutions obtained in this step will contain all solutions to $A(z_1, z_2) = 0$ in the 3-D space of $(|z_1| = 1, z_2)$. If none of these solutions has $|z_2|$ greater than or equal to 1, then $A(z_1, z_2) \neq 0$ for any $(|z_1| = 1, |z_2| \geq 1)$, which satisfies condition (a). Step 2 is clearly a trivial operation. In step 1, we have to solve the following equation:

$$A(e^{j\omega_1}, z_2) = 0$$

Suppose we consider a fixed value of ω_1, say ω_1'. Then $A(e^{j\omega_1'}, z_2)$ is a 1-D polynomial in the variable z_2, and solving for all z_2 such that $A(e^{j\omega_1'}, z_2) = 0$ is equivalent to a 1-D stability test. If we vary ω_1 continuously from 0 to 2π and we perform the 1-D stability test for each ω_1, we will find all possible values of $(e^{j\omega_1}, z_2)$ such that $A(e^{j\omega_1}, z_2) = 0$. In practice, we cannot change ω_1 continuously, and we have to consider discrete values of ω_1. By performing many 1-D stability tests, we can obtain a table such as Table 7.4. By choosing Δ sufficiently small, it is possible to essentially determine all possible values of (ω_1, z_2) such that $A(e^{j\omega_1}, z_2) = 0$. By checking if all the values of $|z_2|$ in Table 7.4 are smaller than 1, we can satisfy condition (a) without a 3-D search.

TABLE 7.4 SOLUTION TO $A(e^{j\omega_1}, z_2) = 0$

ω_1	z_2
0	$a_0, b_0, c_0, d_0, \ldots$
Δ	$a_1, b_1, c_1, d_1, \ldots$
2Δ	$a_2, b_2, c_2, d_2, \ldots$
$\vdots$	$\vdots$
.	
.	
2π	$a_0, b_0, c_0, d_0, \ldots$

The 2-D search problem corresponding to condition (b) can be substituted for by one 1-D stability test. To satisfy condition (b), we need to make sure that $A(z_1, z_2) \neq 0$ in the 2-D space corresponding to $(|z_1| \geq 1, z_2 = 1)$. One approach to satisfy this condition consists of the following two steps.

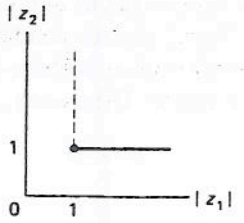
1. Solve all (z_1, z_2) such that $A(z_1, z_2 = 1) = 0$.
2. Check if all $|z_1|$ obtained in step 1 are less than 1.

Step 1 determines all $(z_1, z_2 = 1)$ such that $A(z_1, 1) = 0$. If all $|z_1|$ are less than 1, then $A(z_1, 1)$ cannot be zero for $|z_1| \geq 1$, which satisfies condition (b). Step 2 is a trivial operation. Step 1 is equivalent to a 1-D stability test since $A(z_1, 1)$ is a 1-D polynomial in the variable z_1.

From the preceding discussion, it is clear that a 2-D stability test can be performed by many 1-D stability tests and one 1-D stability test. This fact can help us develop a procedure to be used in practice to test the stability of a 2-D system.

Among the many variations to Huang's theorem is the following:

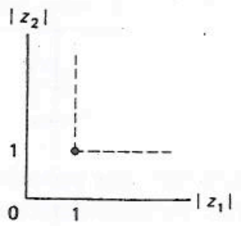
$$\text{Stability} \longleftrightarrow \begin{array}{lll} \text{(a)} & A(z_1, z_2) \neq 0 & \text{for } |z_1| \geq 1,\ |z_2| = 1 \\ \text{(b)} & A(z_1, z_2) \neq 0 & \text{for } z_1 = 1,\ |z_2| \geq 1 \end{array} \tag{7.62}$$



This variation is the same as Huang's theorem except that the roles of z_1 and z_2 have been interchanged.

DeCarlo-Strintzis Theorem. The DeCarlo-Strintzis theorem can be stated as follows:

$$\text{Stability} \longleftrightarrow \begin{array}{lll} \text{(a)} & A(z_1, z_2) \neq 0, & |z_1| = |z_2| = 1 \\ \text{(b)} & A(z_1, 1) \neq 0, & |z_1| \geq 1 \\ \text{(c)} & A(1, z_2) \neq 0, & |z_2| \geq 1 \end{array} \tag{7.63}$$



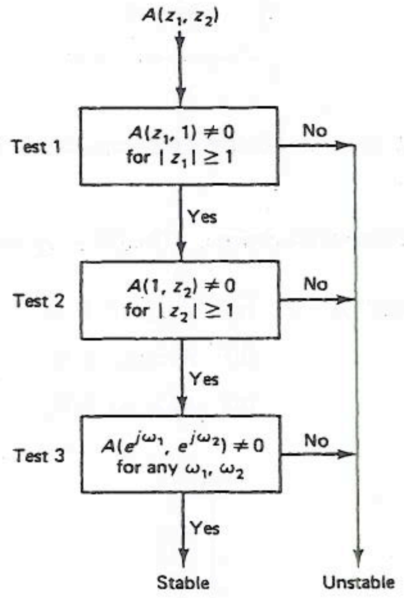
Each of the three conditions in this theorem corresponds to a 2-D search problem. Condition (a) requires $A(z_1, z_2)$ to be nonzero on the 2-D unit surface. Conditions (b) and (c) require $A(z_1, z_2)$ to be nonzero in the 2-D spaces shown by the dotted lines in the figure in Eq. (7.63). From the search point of view, therefore, the conditions imposed by the DeCarlo-Strintzis theorem are considerably simpler than those in Huang's theorem, which requires a 3-D search.

In practice, however, this theorem is not much simpler than Huang's theorem. Specifically, condition (b) in the DeCarlo-Strintzis theorem is the same as condition (b) in Huang's theorem, which can be checked by a 1-D stability test. Condition (c) in the DeCarlo-Strintzis theorem is the same as condition (b) with the roles of z_1 and

z_2 interchanged. Condition (c) can therefore be checked by one 1-D stability test. In the case of condition (a), however, the 2-D search problem cannot be simplified by one 1-D stability test, and a full 2-D search on the unit surface is generally necessary. Computations involved in this 2-D search are often comparable to those of many 1-D stability tests that can be used in testing condition (a) of Huang's theorem.

7.5.3 Methods for Stability Test

From the two theorems discussed in Section 7.5.2, we can develop many different methods to check the stability of a 2-D system. One such method is shown in Fig. 7.36. Test 1 in the figure checks condition (b) of Huang's theorem and the DeCarlo-Strintzis theorem. Test 2 checks condition (c) of the DeCarlo-Strintzis theorem. Test 3 checks condition (a) of the DeCarlo-Strintzis theorem. If a system passes all three tests, the system is stable. Otherwise, it is unstable.



$A(z_1, z_2)$
Test 1
$A(z_1, 1) \neq 0$ for $|z_1| \geq 1$
No
Yes
Test 2
$A(1, z_2) \neq 0$ for $|z_2| \geq 1$
No
Yes
Test 3
$A(e^{j\omega_1}, e^{j\omega_2}) \neq 0$ for any ω_1, ω_2
No
Yes
Stable
Unstable

Figure 7.36 One approach to test the stability of $H(z_1, z_2) = 1/[A(z_1, z_2)]$.

To illustrate how the procedure in Fig. 7.36 can be used in testing the stability of a 2-D system, we consider two examples:

Example 1

$$H(z_1, z_2) = \frac{1}{A(z_1, z_2)} = \frac{1}{1 - 0.6z_1^{-1} - 0.6z_2^{-1}}$$

where $h(n_1, n_2)$ is a first-quadrant sequence.

Test 1

$$A(z_1, 1) = 1 - 0.6z_1^{-1} - 0.6 = 0$$
$$z_1 = \tfrac{3}{2} \geq 1$$

The system fails test 1 and therefore is unstable.

Example 2

$$H(z_1, z_2) = \frac{1}{A(z_1, z_2)} = \frac{1}{1 - \frac{1}{2}z_1^{-1} - \frac{1}{4}z_1^{-1} \cdot z_2^{-1}},$$

where $h(n_1, n_2)$ is a first-quadrant sequence.

Test 1

$$A(z_1, 1) = 1 - \tfrac{1}{2}z_1^{-1} - \tfrac{1}{4}z_1^{-1}$$
$$z_1 = \tfrac{3}{4} < 1$$

Test 1 passed.

Test 2

$$A(1, z_2) = 1 - \tfrac{1}{2} - \tfrac{1}{4} \cdot z_2^{-1}$$
$$z_2 = \tfrac{1}{2} < 1$$

Test 2 passed.

Test 3

$$\begin{aligned} A(\omega_1, \omega_2) &= A(z_1, z_2)\big|_{z_1 = e^{j\omega_1}, z_2 = e^{j\omega_2}} \\ &= 1 - \tfrac{1}{2}e^{-j\omega_1} - \tfrac{1}{4}e^{-j\omega_1} \cdot e^{-j\omega_2} \neq 0 \qquad \text{for any } (\omega_1, \omega_2) \end{aligned}$$

Test 3 passed.

The system passes all three tests and is therefore stable. In this example, test 3 could be performed by inspection. In typical cases, however, test 3 requires a considerable amount of computation.

From the preceding discussion, it is clear that testing the stability of a 2-D system is considerably more complex than testing the stability of a 1-D system. A 2-D stability test problem typically corresponds to many 1-D stability tests. In addition, the stability of a 2-D system cannot in general be absolutely guaranteed by a finite number of 1-D stability tests since $A(e^{j\omega_1}, z_2) = 0$ has to be solved for every possible ω_1. The complexity of testing the stability of a 2-D system and the lack of simple procedures to design a stable filter and to stabilize an unstable filter explain, in part, why 2-D finite impulse response (FIR) digital filters, which are always stable, are much preferred in practice over 2-D infinite impulse response (IIR) digital filters.

7.6 DISCRETE FOURIER TRANSFORM AND FAST FOURIER TRANSFORM

7.6.1 Discrete Fourier Transform (DFT)

In many signal processing applications, such as image processing, we deal with sequences of finite extent. For such sequences, the Fourier transform and z-transform uniformly converge and are well defined. The Fourier transform and z-transform representations $X(\omega_1, \omega_2)$ and $X(z_1, z_2)$ are functions of continuous variables (ω_1, ω_2) and (z_1, z_2). For finite-extent sequences, which can be represented by a finite number of values, then the Fourier transform and z-transform are not computationally convenient frequency-domain representations. The discrete Fourier transform (DFT) is a frequency-domain representation of finite-extent sequences, where a finite-extent sequence is represented in the frequency domain by a finite number of values.

The 2-D DFT representation can be derived by a straightforward extension of the 1-D result. Specifically, the 2-D DFT pair is given by

Discrete Fourier Transform Pair

$$X(k_1, k_2) = \begin{cases} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) \cdot e^{-j(2\pi/N_1)k_1 \cdot n_1} \cdot e^{-j(2\pi/N_2)k_2 \cdot n_2}, \\ \qquad \text{for } 0 \leq k_1 \leq N_1 - 1, \\ \qquad\quad 0 \leq k_2 \leq N_2 - 1 \\ 0, \qquad \text{otherwise} \end{cases} \tag{7.64}$$

$$x(n_1, n_2) = \begin{cases} \frac{1}{N_1 \cdot N_2} \cdot \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) \cdot e^{j(2\pi/N_1)k_1 \cdot n_1} \cdot e^{j(2\pi/N_2)k_2 \cdot n_2}, \\ \qquad \text{for } 0 \leq n_1 \leq N_1 - 1, \\ \qquad\quad 0 \leq n_2 \leq N_2 - 1 \\ 0 \qquad \text{otherwise} \end{cases} \tag{7.65}$$

From Eqs. (7.64) and (7.65), an $N_1 \times N_2$-point sequence $x(n_1, n_2)$ is represented by an $N_1 \times N_2$-point sequence $X(k_1, k_2)$ in the frequency domain. The sequence $X(k_1, k_2)$ is called the DFT of $x(n_1, n_2)$, and $x(n_1, n_2)$ is called the inverse DFT of $X(k_1, k_2)$. The DFT pair given in the box is defined only for a finite-extent first-quadrant sequence. This is not a serious restriction in practice since a finite-extent sequence can always be shifted, and the shift can easily be accounted for in typical applications.

For a finite-extent first-quadrant sequence $x(n_1, n_2)$ that is zero outside $0 \leq n_1 \leq N_1 - 1$ and $0 \leq n_2 \leq N_2 - 1$, the DFT $X(k_1, k_2)$ is related in a straightforward way to the discrete-space Fourier transform $X(\omega_1, \omega_2)$. From Eqs. (7.28) and (7.64), it is clear that

$$X(k_1, k_2) = X(\omega_1, \omega_2)\big|_{\omega_1=(2\pi/N_1)k_1,\, \omega_2=(2\pi/N_2)k_2} \qquad \text{for } 0 \leq n_1 \leq N_1 - 1,\ 0 \leq n_2 \leq N_2 - 1 \tag{7.66}$$

Equation (7.66) states that the DFT coefficients of $x(n_1, n_2)$ are samples of $X(\omega_1, \omega_2)$ at equally spaced points on the Cartesian grid, beginning at $\omega_1 = \omega_2 = 0$. Since $X(k_1, k_2)$ completely specifies $x(n_1, n_2)$, which in turn completely specifies $X(\omega_1, \omega_2)$, $X(\omega_1, \omega_2)$ has considerable redundant information; e.g., $N_1 \times N_2$ samples of $X(\omega_1, \omega_2)$ completely specify $X(\omega_1, \omega_2)$.

We can derive a number of useful properties from the DFT pair of equations (7.64) and (7.65). Some of the important properties, often useful in practice, are listed in Table 7.5. Most of these properties are straightforward extensions of 1-D results.

Analogous to the 1-D case, property 2 and property 4 present alternative ways to perform linear convolution of two finite-extent sequences. To linearly convolve $f(n_1, n_2)$ and $g(n_1, n_2)$, we could assume the proper periodicity $N_1 \times N_2$, determine

TABLE 7.5 PROPERTIES OF THE DISCRETE FOURIER TRANSFORM

$$x(n_1, n_2), y(n_1, n_2) = 0 \quad \text{outside } 0 \le n_1 \le N_1 - 1,\ 0 \le n_2 \le N_2 - 1$$

$$x(n_1, n_2) \longleftrightarrow X(k_1, k_2) \qquad y(n_1, n_2) \longleftrightarrow Y(k_1, k_2)$$

$$R_{N_1 \times N_2}(n_1, n_2) = \begin{cases} 1, & 0 \le n_1 \le N_1 - 1,\ 0 \le n_2 \le N_2 - 1 \\ 0, & \text{otherwise} \end{cases}$$

$N_1 \times N_2$-point DFT and inverse DFT are assumed.

Property 1: Linearity

$$a \cdot x(n_1, n_2) + b \cdot y(n_1, n_2) \longleftrightarrow a \cdot X(k_1, k_2) + b \cdot Y(k_1, k_2)$$

Property 2: Circular convolution

$$x(n_1, n_2) \circledast y(n_1, n_2) \longleftrightarrow X(k_1, k_2) \cdot Y(k_1, k_2)$$

Property 3: Separable sequence

$$x(n_1, n_2) = x_1(n_1) \cdot x_2(n_2) \longleftrightarrow X(k_1, k_2) = X_1(k_1) \cdot X_2(k_2),$$
$$X_1(k_1)\text{: } N_1\text{-point 1-D DFT}, \qquad X_2(k_2)\text{: } N_2\text{-point 1-D DFT}$$

Property 4: Relation between circular and linear convolution

$$f(n_1, n_2) = 0 \quad \text{outside } 0 \le n_1 \le N_1' - 1,\ 0 \le n_2 \le N_2' - 1$$
$$g(n_1, n_2) = 0 \quad \text{outside } 0 \le n_1 \le N_1'' - 1,\ 0 \le n_2 \le N_2'' - 1$$
$$f(n_1, n_2) * g(n_1, n_2) = f(n_1, n_2) \circledast g(n_1, n_2)$$
with periodicity $N_1 \ge N_1' + N_1'' - 1,\ N_2 \ge N_2' + N_2'' - 1$

$F(k_1, k_2)$ and $G(k_1, k_2)$, multiply $F(k_1, k_2)$ and $G(k_1, k_2)$, and then perform the inverse DFT operation of $F(k_1, k_2) \cdot G(k_1, k_2)$. Even though this approach appears to be quite cumbersome, it sometimes reduces the amount of computations involved in performing linear convolution in practical applications. The 1-D methods for performing convolution such as the overlap-add method and the overlap-save method, which are based on this notion of performing convolution by computing DFTs, extend to the problem of performing 2-D convolution in a straightforward manner.

7.6.2 Fast Fourier Transform (FFT)

Row-column decomposition. The DFT discussed in previous sections is used in a variety of signal processing applications, so it is of considerable interest to efficiently compute the DFT and inverse DFT. One efficient way to compute the 2-D DFT is known as the fast Fourier transform by row-column decomposition. This simple method uses 1-D FFT algorithms and offers considerable computational savings compared with direct computation. It is also the most popular 2-D FFT algorithm.

To appreciate the computational efficiency of the row-column decomposition method, we first consider computing the DFT directly (the inverse DFT computation is essentially the same). Consider an $N_1 \times N_2$-point complex sequence $x(n_1, n_2)$ that is zero outside $0 \le n_1 \le N_1 - 1$, $0 \le n_2 \le N_2 - 1$. The DFT of $x(n_1, n_2)$, $X(k_1, k_2)$, is related to $x(n_1, n_2)$ by

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) \cdot e^{-j(2\pi/N_1)k_1 \cdot n_1} \cdot e^{-j(2\pi/N_2)k_2 \cdot n_2}, \quad 0 \le k_1 \le N_1 - 1,\ 0 \le k_2 \le N_2 - 1 \tag{7.67}$$

From Eq. (7.67), directly computing $X(k_1, k_2)$ for each (k_1, k_2) requires $N_1 \cdot N_2 - 1$ complex additions and $N_1 \cdot N_2$ complex multiplications. Since there are $N_1 \cdot N_2$ different values of (k_1, k_2), the total number of arithmetic operations required in computing $X(k_1, k_2)$ from $x(n_1, n_2)$ is $N_1^2 \cdot N_2^2$ complex multiplications and $N_1 \cdot N_2 \cdot (N_1 \cdot N_2 - 1)$ complex additions.

To develop the row-column decomposition method, we rewrite Eq. (7.67) as follows:

$$X(k_1, k_2) = \sum_{n_2=0}^{N_2-1} \underbrace{\sum_{n_1=0}^{N_1-1} x(n_1, n_2) \cdot e^{-j(2\pi/N_1)k_1 \cdot n_1}}_{f(k_1, n_2)} \cdot e^{-j(2\pi/N_2)k_2 \cdot n_2}, \quad 0 \le k_1 \le N_1 - 1,\ 0 \le k_2 \le N_2 - 1 \tag{7.68}$$

Consider a fixed n_2, say $n_2 = 0$. Then $x(n_1, n_2)|_{n_2=0}$ represents a row of $x(n_1, n_2)$, and $f(k_1, n_2)|_{n_2=0}$ is nothing but the 1-D N_1-point DFT of $x(n_1, n_2)|_{n_2=0}$ with respect to the variable n_1. Therefore, we can compute $f(k_1, 0)$ from $x(n_1, n_2)$ by computing one 1-D N_1-point DFT. Since there are N_2 different values for n_2 in $f(k_1, n_2)$ that are of interest to us, we can compute $f(k_1, n_2)$ from $x(n_1, n_2)$ by computing N_2 1-D N_1-point DFTs. This process is illustrated in Fig. 7.37.

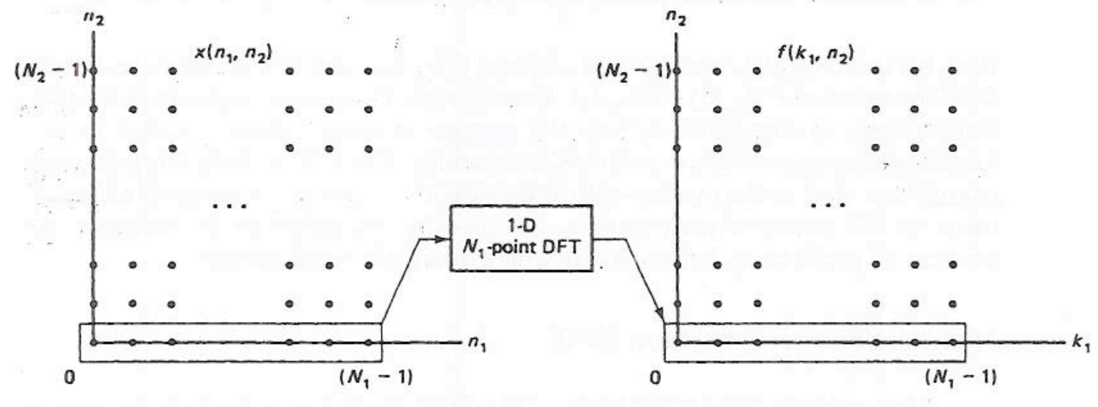


Figure 7.37 Computation of $f(k_1, n_2)$ from $x(n_1, n_2)$ by performing N_2 1-D N_1-point DFTs.

Once we compute $f(k_1, n_2)$ from Eq. (7.68), we can compute $X(k_1, k_2)$ from $f(k_1, n_2)$ as follows:

$$X(k_1, k_2) = \sum_{n_2=0}^{N_2-1} f(k_1, n_2) \cdot e^{-j(2\pi/N_2)k_2 \cdot n_2}, \quad 0 \le k_1 \le N_1 - 1,\ 0 \le k_2 \le N_2 - 1 \tag{7.69}$$

To compute $X(k_1, k_2)$ from $f(k_1, n_2)$, consider a fixed k_1, say $k_1 = 0$. Then $f(k_1, n_2)|_{k_1=0}$ represents one column of $f(k_1, n_2)$, and $X(k_1, k_2)|_{k_1=0}$ in Eq. (7.69) is nothing but the 1-D N_2-point DFT of $f(k_1, n_2)|_{k_1=0}$ with respect to the variable n_2. Therefore, we can compute $X(0, k_2)$ from $f(k_1, n_2)$ by computing one 1-D N_2-point DFT. Since there are N_1 different values for k_1 in $X(k_1, k_2)$ that are of interest to us, we can compute $X(k_1, k_2)$ from $x(n_1, n_2)$ by computing N_1 1-D N_2-point DFTs. This process is illustrated in Fig. 7.38.

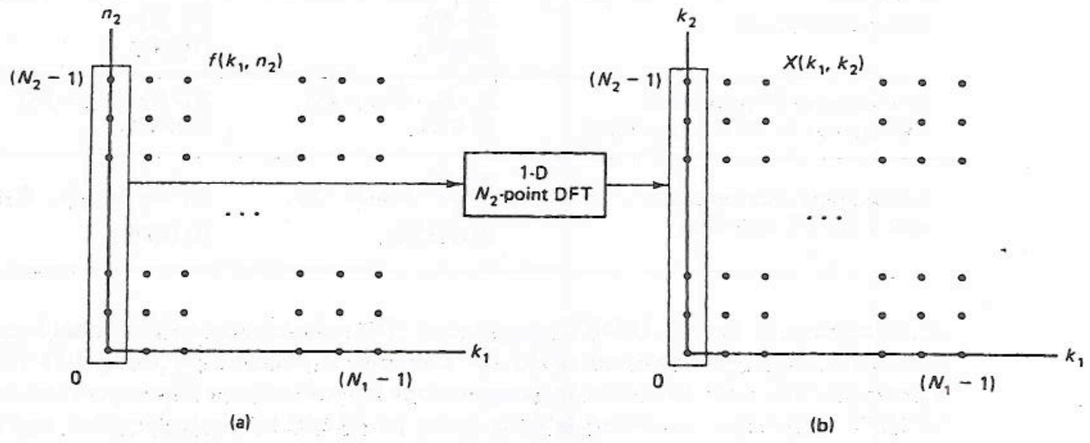


Figure 7.38 Computation of $X(k_1, k_2)$ from $f(k_1, n_2)$ by performing N_1 1-D N_2-point DFTs.

From the preceding discussion, we can compute $X(k_1, k_2)$ from $x(n_1, n_2)$ with a total of N_2 1-D N_1-point DFTs for the row operations and N_1 1-D N_2-point DFTs for the column operations. Suppose we compute the 1-D DFTs directly. Since direct computation of one 1-D N-point DFT requires N^2 multiplications and about N^2 additions, the total number of arithmetic operations involved in computing $X(k_1, k_2)$ is $N_1 \cdot N_2(N_1 + N_2)$ multiplications and $N_1 \cdot N_2(N_1 + N_2)$ additions. This is a significant computational saving relative to the $N_1^2 \cdot N_2^2$ multiplications and $N_1^2 \cdot N_2^2$ additions required for direct computation of $X(k_1, k_2)$.

To further reduce the number of arithmetic operations, we can of course use 1-D FFT algorithms to compute the 1-D DFTs in the preceding discussion. When $N = 2^M$, an N-point 1-D FFT algorithm requires $(N/2) \cdot \log_2^N$ multiplications and $N \cdot \log_2^N$ additions. To compute N_2 1-D N_1-point DFTs and N_1 1-D N_2-point DFTs using 1-D FFT algorithms when $N_1 = 2^{M_1}$ and $N_2 = 2^{M_2}$, we need a total of $[(N_1 \cdot N_2)/2] \log_2 N_1 \cdot N_2$ multiplications and $N_1 \cdot N_2 \log_2 N_1 \cdot N_2$ additions. This is a significant computational saving relative to direct computation of the 1-D DFTs. If we represent the total number of points $N_1 \cdot N_2$ as N, then the number of computations involved in the preceding case can be expressed as $(N/2) \cdot \log_2^N$ multiplications and $N \cdot \log_2^N$ additions. These are exactly the same expressions as those for the 1-D N-point DFT computations, using an FFT algorithm such as a decimation-in-time algorithm.

To appreciate the computational saving involved, Table 7.6 shows the relative amount of computations for the three methods. When $N_1 = N_2 = 512$, row-column decomposition alone reduces the number of multiplications and additions by a factor

TABLE 7.6 COMPARISON OF THREE METHODS TO COMPUTE A 2-D $N_1 \times N_2$-POINT DFT IN THE REQUIRED NUMBER OF MULTIPLICATIONS AND ADDITIONS

	Number of multiplications ($N_1 = N_2 = 512$)	Number of additions ($N_1 = N_2 = 512$)
Direct computation	$N_1^2 \cdot N_2^2$ (100%)	$N_1^2 \cdot N_2^2$ (100%)
Row-column decomposition with direct 1-D DFT composition	$N_1 \cdot N_2 \cdot (N_1 + N_2)$ (0.4%)	$N_1 \cdot N_2 \cdot (N_1 + N_2)$ (0.4%)
Row-column decomposition with 1-D FFT algorithm	$\frac{N_1 \cdot N_2}{2} \cdot \log_2 N_1 \cdot N_2$ (0.0035%)	$N_1 \cdot N_2 \cdot \log_2(N_1 \cdot N_2)$ (0.007%)

of 250 relative to direct 2-D DFT computation. The reduction by an additional factor of 110 for multiplications and of 55 for additions is obtained by using 1-D FFT algorithms. The total reduction in computations by row-column decomposition and 1-D FFT algorithms combined is by a factor of 30,000 for multiplications and of 15,000 for additions for $N_1 = N_2 = 512$ compared with the direct 2-D DFT computation.

In the derivation of the row-column decomposition approach, we expressed $X(k_1, k_2)$ in the form of Eq. (7.68). This led to the procedure where we performed row operations before column operations. An alternative way to write Eq. (7.68) is as follows:

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \underbrace{\sum_{n_2=0}^{N_2-1} x(n_1, n_2) \cdot e^{-j(2\pi/N_2)k_2 \cdot n_2}}_{g(n_1, k_2)} \cdot e^{-j(2\pi/N_1)k_1 \cdot n_1}, \qquad \begin{array}{l} 0 \le k_1 \le N_1 - 1, \\ 0 \le k_2 \le N_2 - 1 \end{array} \tag{7.70}$$

If Eq. (7.70) is used in computing $X(k_1, k_2)$, then the column operations are performed before row operations. The computations involved in this case remain the same as in the previous case, where the row operations are performed first.

The computation of a reasonable-size 2-D DFT requires a fair amount of memory. When we compute the 2-D DFT of size 512×512, we need about a quarter of a million memory locations. As the cost of memory becomes cheaper, this amount of memory probably will not be a major issue in practical applications. If the memory size is an important consideration, however, the data may have to be stored on slow memory such as disk memory, and the number of I/O operations must be reduced. If rows of the data $x(n_1, n_2)$ are stored as blocks of data on the disk, one efficient approach is to perform the row operations first, transpose the result so that columns become rows, and then perform row operations on the transposed result. The result can then be transposed to have $X(k_1, k_2)$ stored in the proper order. Procedures for data transposition that are efficient in the required number of I/O operations can be found in [1].

The row-column decomposition approach discussed in this section is very efficient computationally, is conceptually simple to understand, and can be implemented using existing 1-D FFT algorithms. For these reasons, it is probably the most popular method used in the 2-D DFT computation.

Vector radix FFT. In the row-column decomposition method, the 2-D DFT computation is transformed to many 1-D DFT computations, and 1-D FFT algorithms are used efficiently to compute the 1-D DFTs. An alternative approach is to extend the idea behind 1-D FFT algorithm development directly to the 2-D case. This extension leads to vector radix FFT algorithms.

Although there are many variations, all 1-D FFT algorithms are based on one simple principle: an N-point DFT can be computed by two $(N/2)$-point DFTs or three $(N/3)$-point DFTs, etc. This simple principle can be extended to the 2-D case in a straightforward manner. Specifically, an $N_1 \times N_2$-point DFT can be computed by four $(N_1/2) \times (N_2/2)$-point DFTs, or six $(N_1/2) \times (N_2/3)$-point DFTs, or nine $(N_1/3) \times (N_2/3)$-point DFTs, etc. Using this extension, various 1-D FFT algorithms such as decimation-in-time and decimation-in-frequency algorithms can be directly extended to the 2-D case, and all the properties also extend directly from the 1-D case to the 2-D case. In the 2-D decimation-in-space algorithm, for example, in-place computation is possible, and bit reversal of input is necessary to have correct output and in-place computation. Compared with the row-column decomposition method, vector radix FFT algorithms are roughly the same in various aspects and do not offer any significant advantages. The amount of computations and memory locations required, for example, are roughly the same in both cases.

7.7 FINITE IMPULSE RESPONSE DIGITAL FILTERS

Three steps are generally followed in using digital filters. In the first step, we specify the characteristics required of the filter. The filter specification depends, of course, on the application for which the filter is used. For example, in a case where we wish to restore a signal that has been degraded by background noise, the filter characteristics we require depend on the spectral characteristics of the signal and the background noise. The second step is the filter design step, where we determine $h(n_1, n_2)$, the unit sample response of the filter, or its system function $H(z_1, z_2)$ that meets the design specification. The third step is the filter implementation step, in which we realize a discrete space system with the given $h(n_1, n_2)$ or $H(z_1, z_2)$.

The three steps are closely related. For example, it does not make much sense to specify a filter that cannot be designed. Neither does it make much sense to design a filter that cannot be implemented. Despite the close relationship among the three steps, we discuss them separately and point out the interrelationships as appropriate.

We restrict ourselves to a certain class of digital filters for practical reasons. One restriction is that $h(n_1, n_2)$ be real. In practice, we often deal with real data. So to ensure that the processed signal is real when the input signal is real, we require $h(n_1, n_2)$ to be real. Another restriction is the stability of $h(n_1, n_2)$, i.e., $\sum_{n=-\infty}^{\infty} |h(n_1, n_2)| < \infty$. In practice, an unbounded output can cause many difficulties such as system overload. For these practical reasons, we restrict our discussion to the class of digital filters whose unit sample response $h(n_1, n_2)$ is real and stable.

Digital filters can often be classified into two groups. In the first group $h(n_1, n_2)$ is a finite-extent sequence, and the filters in this group are called finite impulse response (FIR) filters. In the second group, $h(n_1, n_2)$ is of infinite extent, and the filters in this group are called infinite impulse response (IIR) filters. In this section we concentrate on FIR filters and in Section 7.8 on IIR filters. As in the 1-D case, the design and implementation of FIR filters differ considerably from those of IIR filters.

7.7.1 Zero-Phase Filters

A digital filter $h(n_1, n_2)$ is said to have zero phase when its frequency response $H(\omega_1, \omega_2)$ is a real function, so that

$$H(\omega_1, \omega_2) = H^*(\omega_1, \omega_2) \tag{7.71}$$

Strictly speaking, the filter whose frequency response is real may not be a zero-phase filter, since $H(\omega_1, \omega_2)$ can be negative. In practice, the frequency regions for which $H(\omega_1, \omega_2)$ is negative typically correspond to the stopband regions, and a phase of 180° in the stopband regions has little significance.

From the symmetry properties of the Fourier transform, Eq. (7.71) is equivalent in the spatial domain to the following expression:

$$h(n_1, n_2) = h^*(-n_1, -n_2) \tag{7.72}$$

Since we consider only real $h(n_1, n_2)$, Eq. (7.72) reduces to

$$h(n_1, n_2) = h(-n_1, -n_2) \tag{7.73}$$

Equation (7.73) states that the unit sample response of a zero-phase filter is symmetric with respect to the origin.

One characteristic of a zero-phase filter is its tendency to preserve the shape of the signal component in the passband region of the filter. This characteristic is quite useful in applications such as image processing, where the shape of the signal is very important. In addition, from Eq. (7.73) it is very easy to require zero phase for FIR filters, and design and implementation are often simplified if we require zero phase. For these reasons, we restrict our discussion of FIR filters to zero-phase filters.

7.7.2 Filter Specification

Like 1-D digital filters, 2-D digital filters are generally specified in the frequency domain. Since $H(\omega_1, \omega_2) = H(\omega_1 + 2\pi, \omega_2) = H(\omega_1, \omega_2 + 2\pi)$ for all (ω_1, ω_2), $H(\omega_1, \omega_2)$ for $-\pi \le \omega_1, \omega_2 \le \pi$ completely specifies $H(\omega_1, \omega_2)$. In addition, since $h(n_1, n_2)$ is assumed real, $H(\omega_1, \omega_2) = H^*(-\omega_1, -\omega_2)$. Specifying $H(\omega_1, \omega_2)$ for $-\pi \le \omega_1 \le \pi$, $0 \le \omega_2 \le \pi$ therefore completely specifies $H(\omega_1, \omega_2)$ for all (ω_1, ω_2).

Since $H(\omega_1, \omega_2)$ is in general a complex function of (ω_1, ω_2), we need to specify both the magnitude and the phase of $H(\omega_1, \omega_2)$. For FIR filters, we require zero phase and therefore need to specify only the magnitude response. Like the 1-D filter specification, one scheme that is sometimes used for the magnitude specification is a "tolerance" scheme. An example of a lowpass filter specified using a tolerance scheme is shown in Fig. 7.39. The filter has a passband region where we require $1 - \delta_p \leq |H(\omega_1, \omega_2)| \leq 1 + \delta_p$ and a stopband region where we require $|H(\omega_1, \omega_2)| \leq \delta_s$. The variables δ_p and δ_s are "passband tolerance" and "stopband tolerance" respectively. Other filters can also be specified analogously.

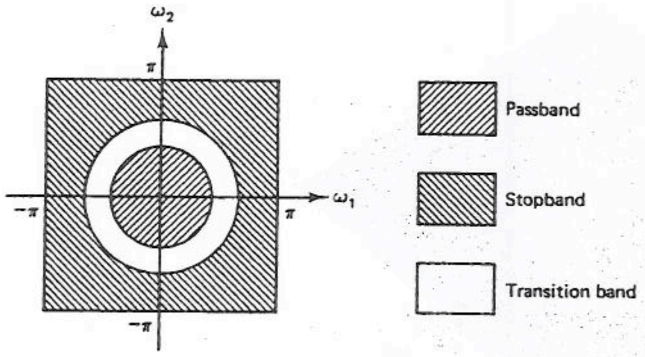


Figure 7.39 An example of a 2-D lowpass filter specification using a tolerance scheme. In the passband region, $1 - \delta_p \leq H(\omega_1, \omega_2) \leq 1 + \delta_p$. In the stopband region, $H(\omega_1, \omega_2) \leq \delta_s$.

7.7.3 FIR Filter Design

The problem of designing a filter is basically that of determining $h(n_1, n_2)$ or $H(z_1, z_2)$ that meets the design specification. The four standard approaches to designing FIR filters are the window method, frequency sampling method, the optimal filter design, and the transformation method. The window method and frequency sampling method are straightforward extensions of 1-D results. The optimal design problem differs significantly between the 1-D and the 2-D case. In the 1-D case, practical algorithms exist for the design of optimal filters. In the 2-D case, practical algorithms for designing optimal filters have not yet been developed. In the transformation method, a 2-D filter is designed from a 1-D filter. There is no counterpart of this method in the design of 1-D FIR filters. We discuss each of the four methods, with much greater emphasis on the methods with significant differences between the 1-D and the 2-D case.

Window method. The window method is a straightforward extension of the 1-D results. In the window method, the desired frequency response $H_d(\omega_1, \omega_2)$ is assumed to be known. By performing an inverse Fourier transform on $H_d(\omega_1, \omega_2)$, we can determine the desired unit sample response of the filter, $h_d(n_1, n_2)$. In general $h_d(n_1, n_2)$ is an infinite-extent sequence. In the window method, we obtain an FIR filter by applying a window $w(n_1, n_2)$ to $h_d(n_1, n_2)$. If $h_d(n_1, n_2)$ and $w(n_1, n_2)$ are symmetric with respect to the origin, then $h_d(n_1, n_2) \cdot w(n_1, n_2)$ is also symmetric with respect to the origin, and therefore the resulting filter is zero phase.

A 2-D window used in the filter design is typically obtained from a 1-D window by using one of the following two methods:

$$w(n_1, n_2) = w_a(t_1) \cdot w_b(t_2)\big|_{t_1=n_1, t_2=n_2} \quad \text{or} \tag{7.74}$$

$$w(n_1, n_2) = w_a(t)\big|_{t=\sqrt{n_1^2+n_2^2}} \tag{7.75}$$

where $w_a(t)$ and $w_b(t)$ are 1-D analog windows. Equation (7.74) leads to a separable rectangularly shaped window, and Eq. (7.75) leads to a circularly symmetric window. The shape and effective size of the 1-D windows used in Eqs. (7.74) and (7.75) are determined by recognizing that the sidelobe behavior and therefore the passband and stopband tolerances are affected primarily by the window shape only and that the

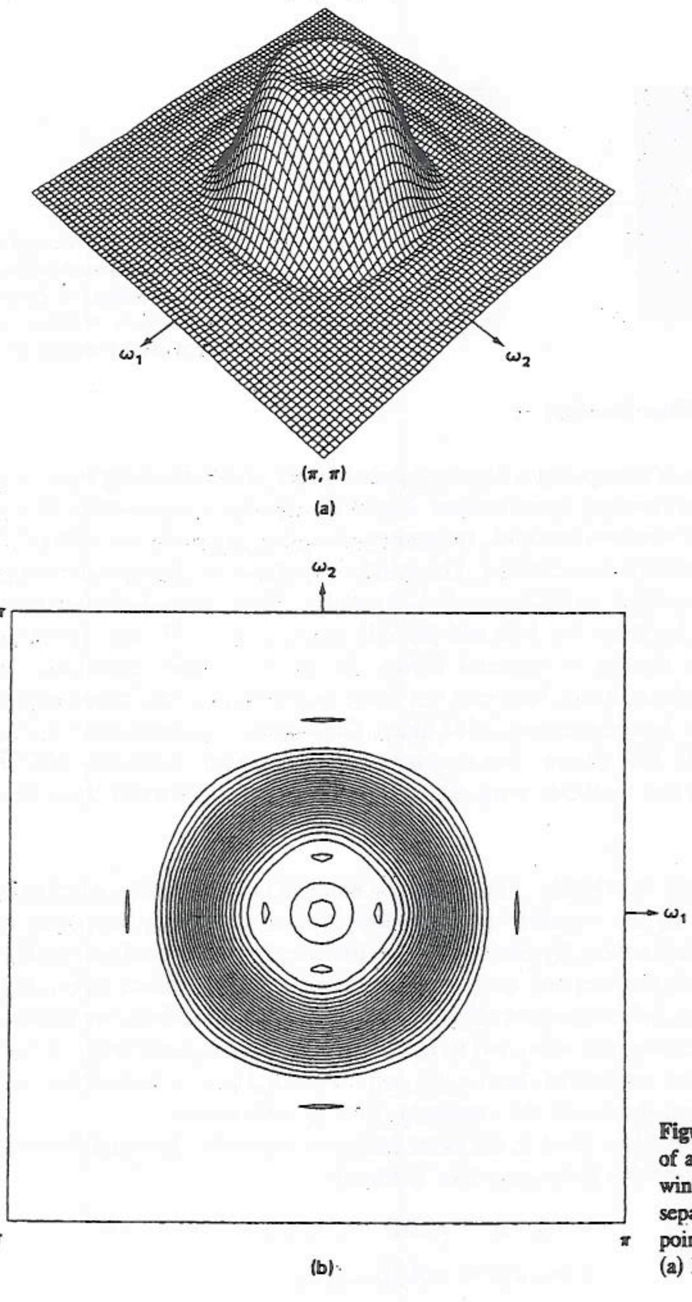


Figure 7.40 The frequency response of a lowpass filter designed by the window method. The window is a separable Kaiser window, of 9×9 points with $\omega_c = 0.4\pi$ in Eq. (7.78). (a) Perspective plot; (b) contour plot.

mainlobe behavior and therefore the transition width are affected by both the window shape and the effective window size. In a typical design, therefore, the window shape is chosen first based on the passband and stopband tolerance requirements and then the window size is determined based on the transition width requirements. Two examples of digital filters designed by the window method are shown in Figs. 7.40 and 7.41.

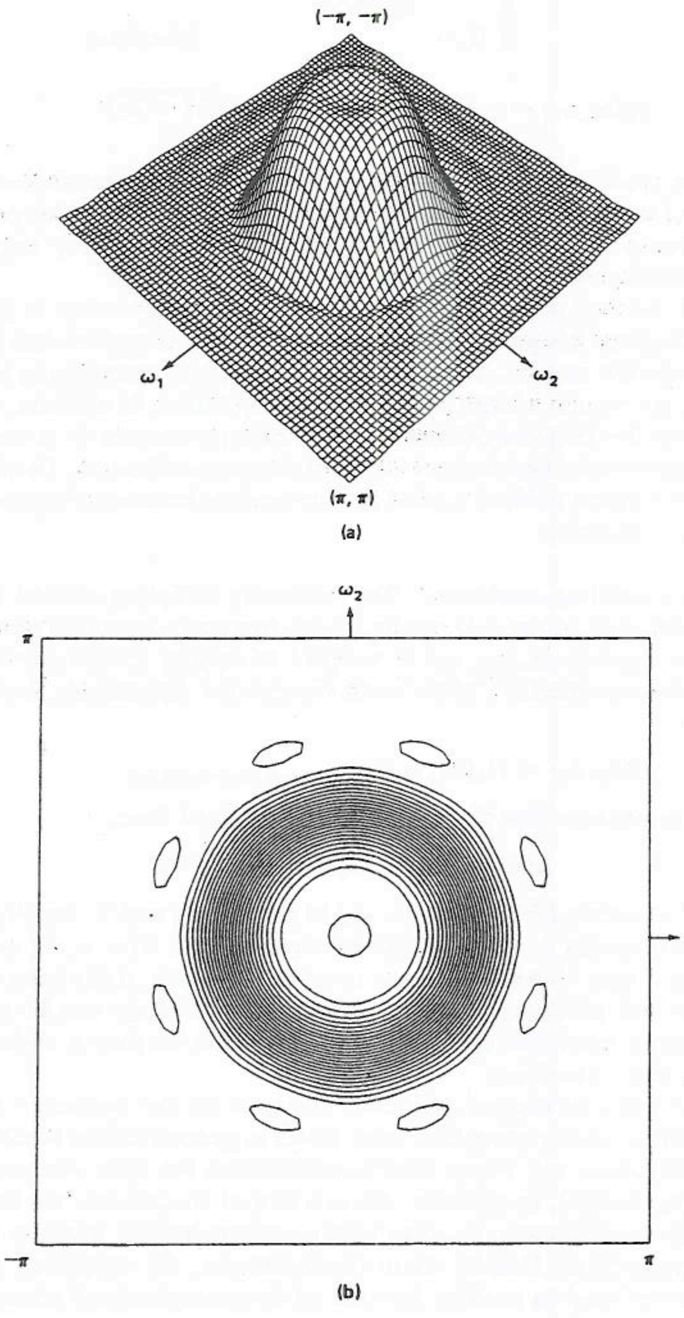


Figure 7.41 The frequency response of a lowpass filter designed by the window method. The window is a circularly symmetric Kaiser window, enclosed in a square of 9×9 points, with $\omega_c = 0.4\pi$ in Eq. (7.78). (a) Perspective plot; (b) contour plot.

In Fig. 7.40, Eq. (7.74) was used, and in Fig. 7.41, Eq. (7.75) was used. In both cases, the 1-D analog Kaiser window and $h_d(n_1, n_2)$ used are

$$w_a(t) = w_b(t) = \begin{cases} \dfrac{I_0\left(0.4\sqrt{1-\dfrac{t^2}{N^2}}\right)}{I_0(0.4)}, & t \leq N \\ 0, & \text{otherwise} \end{cases} \tag{7.76}$$

$$h_d(n_1, n_2) = \frac{\omega_c}{2\pi\sqrt{n_1^2+n_2^2}} \cdot J_1(\omega_c \cdot \sqrt{n_1^2+n_2^2}) \tag{7.77}$$

where $I_0(x)$ is the modified Bessel function of the first kind of zeroth order. The sequence $h_d(n_1, n_2)$ used corresponds to the circularly symmetric ideal lowpass filter with cutoff frequency $\omega_c = 0.4\pi$. In each type, the filter frequency response is displayed by a perspective plot and contour plot.

The window method is not optimal, in the sense that there exists in general a filter that meets the given design specification and whose size is smaller than the filter designed by the window method. For an arbitrary $H_d(\omega_1, \omega_2)$, determining $h_d(n_1, n_2)$ from $H_d(\omega_1, \omega_2)$ may require a large inverse DFT computation. In addition, due to a lack of control over the frequency-domain specification parameters, it is sometimes necessary to design several filters to meet the given design specification. Despite these disadvantages, the window method is often used in practice because of its conceptual and computational simplicity.

Frequency sampling method. The frequency sampling method is also a straightforward extension of the 1-D results. In the frequency sampling method, the desired frequency response $H_d(\omega_1, \omega_2)$ is sampled at equally spaced points on the Cartesian grid and the inverse DFT of the result is computed. Specifically, let $H(k_1, k_2)$ be obtained by

$$H(k_1, k_2) = H_d(\omega_1, \omega_2)\big|_{\omega_1=(2\pi/N_1)k_1,\, \omega_2=(2\pi/N_2)k_2} \tag{7.78}$$

The unit sample response of the filter, $h(n_1, n_2)$, is obtained from

$$h(n_1, n_2) = \text{IDFT}[H(k_1, k_2)] \tag{7.79}$$

where IDFT is the inverse DFT. If $H_d(\omega_1, \omega_2)$ is zero phase and N_1 and N_2 are odd integers, then the resulting $h(n_1, n_2)$ is also zero phase, i.e., $h(n_1, n_2)$ is symmetric with respect to the origin. When $H_d(\omega_1, \omega_2)$ is sampled exactly, it has been observed that the stopband and passband behavior are rather poor. They can be improved considerably if some transition samples are taken in the frequency region where $H_d(\omega_1, \omega_2)$ has a sharp transition.

As with the window method, the filter designed by the frequency sampling method is not optimal, in the sense that there exists in general a filter that meets the same design specification and whose size is smaller than the filter designed by the frequency sampling method. In addition, due to a lack of control over the frequency-domain specification parameters, we may have to design several filters to meet the given design specification. Despite these disadvantages, the frequency sampling method is sometimes used in practice because of its conceptual and computational simplicity. Determining specific values and the region of the transition samples is more cumbersome than with the window method, but an inverse transform of $H_d(\omega_1, \omega_2)$ is not needed in the frequency sampling method. Performance, measured in terms of the filter size needed to meet a given design specification, appears to be similar for both the window method and the frequency sampling method.

Optimal filter design. Unlike the 1-D case, no practical procedures have been developed to reliably design a 2-D optimal FIR filter. A detailed discussion of the 2-D optimal FIR filter design problem requires considerable effort, so we only contrast some major differences between the 1-D and 2-D cases to suggest the complexity of the 2-D case relative to the 1-D case.

We first review briefly the 1-D optimal filter design problem. For simplicity, we concentrate on the design of a zero-phase lowpass filter with design specification parameters δ_p (passband tolerance), δ_s (stopband tolerance), ω_p (passband frequency), and ω_s (stopband frequency). The problem of designing an optimal filter can be stated as follows.

Problem 1: Given ω_p, ω_s, $k = \delta_p/\delta_s$, and N (filter length), determine $h(n)$ such that the design specification is satisfied with the smallest δ_s.

This problem can be shown to be a special case of a weighted Chebyshev approximation problem, which is a functional approximation problem. The weighted Chebyshev approximation problem has been studied extensively in mathematics. One theorem, the alternation theorem, states that the problem has a unique solution. The theorem provides a necessary and sufficient condition for the unique solution. The Remez multiple exchange algorithm exploits this necessary and sufficient condition to solve the weighted Chebyshev approximation problem. The Remez exchange algorithm was first used by Parks and McClellan [2] to solve the optimal filter design problem stated as Problem 1. The optimal filter design algorithm based on the Remez exchange algorithm is an iterative procedure in which the filter is improved in each iteration. Each iteration consists of two steps. One step is the determination of candidate filter coefficients $h(n)$ from candidate "alternation frequencies," which involves solving a set of linear equations. The other step is the determination of candidate alternation frequencies from the candidate filter coefficients. This step involves evaluating $H(\omega)$ on a dense grid of ω and looking for local extrema of $H(\omega)$. Once the local extrema are found, candidate alternation frequencies can be determined straightforwardly from the local extrema and bandedge frequencies (0, π, ω_p, and ω_s). The iterative algorithm is guaranteed to converge to the desired solution. Experience has shown that the algorithm converges very fast, and it is widely used in practice to design optimal filters.

A 2-D zero-phase optimal lowpass filter corresponding to Problem 1 can be stated as follows.

Problem 2: Given R_p (passband region), R_s (stopband region), $k = \delta_p/\delta_s$, R_h (support region of $h(n_1, n_2)$), determine $h(n_1, n_2)$ such that the design specification is satisfied with the smallest δ_s.

To solve this problem, an approach similar to the 1-D case has been considered. A theorem, analogous to the alternation theorem, applies to Problem 2. Unlike the 1-D case, the theorem states that the problem does not have a unique solution. This is not much of an issue since we can obtain any one of the many possible solutions. The theorem also provides a necessary and sufficient condition for the solutions to the problem. An iterative algorithm similar to the Remez multiple exchange algorithm exploits this necessary and sufficient condition. As in the 1-D case, the iterative algorithm attempts to improve the filter in each iteration. Each iteration consists of two steps. One step is the determination of candidate filter coefficients $h(n_1, n_2)$ from candidate "critical-point frequencies," which are analogous to alternation frequencies. This step involves solving a set of linear equations. The other step is the determination of candidate critical-point frequencies from the candidate filter coefficients. This step involves evaluating $H(\omega_1, \omega_2)$ on a dense grid of (ω_1, ω_2) and looking for local extrema of $H(\omega_1, \omega_2)$. Compared with the 1-D case, evaluation of $H(\omega_1, \omega_2)$ on a dense grid requires computations that are typically several orders of magnitude greater than required by evaluation of $H(\omega)$. In addition, finding local extrema of $H(\omega_1, \omega_2)$ requires searching the function along many directions at many frequencies, while finding the local extrema of $H(\omega)$ requires searching the function only in one direction. Once the local extrema are found, candidate critical-point frequencies are determined from the local extrema. In the 1-D case, this is straightforward since all local extrema, with a possible exception of $\omega = 0, \pi$, are alternation frequencies. In the 2-D case, however, choosing a set of frequencies that form a critical set from the local extrema is quite involved and complex. Partly because of the difficulties already cited, the iterative algorithm developed so far is very expensive computationally, has not been demonstrated to converge to a correct solution, and is seldom used in practice. Developing a computationally efficient algorithm for the 2-D optimal FIR filter remains as an area for further research.

Transformation method. In the transformation method, a 2-D zero-phase FIR filter is designed from 1-D zero-phase FIR filter. To illustrate the basic idea, consider the following transformation:

$$H(\omega_1, \omega_2) = H(\omega)\big|_{\omega=G(\omega_1, \omega_2)} \tag{7.80}$$

where $H(\omega)$ is a 1-D digital filter frequency response and $H(\omega_1, \omega_2)$ is the frequency response of the resulting 2-D digital filter. Suppose $H(\omega)$ is a bandpass filter as shown in Fig. 7.42. Consider one particular frequency, $\omega = \omega_0'$. Suppose the function $\omega_0' = G(\omega_1, \omega_2)$ represents a contour in the (ω_1, ω_2)-plane shown in Fig. 7.42. Then, according to Eq. (7.80), $H(\omega_1, \omega_2)$ evaluated on the contour equals $H(\omega_0')$. If we now consider other frequencies $\omega_1', \omega_2', \ldots, \omega_N'$ and if their corresponding contours are as shown in Fig. 7.42, then the resulting 2-D filter will be a bandpass filter.

Several important issues need to be considered in this method. One issue is whether or not the resulting 2-D filter is a zero-phase FIR filter. The second issue is whether or not a transformation function $G(\omega_1, \omega_2)$ exists such that there will be a nice mapping between ω and $\omega = G(\omega_1, \omega_2)$ such as shown in Fig. 7.42. Both of these issues are resolved by using a 1-D zero-phase filter and the appropriate transformation function.

Consider a 1-D zero-phase FIR filter $h(n)$ with length $2N + 1$. The frequency response $H(\omega)$ can be expressed as

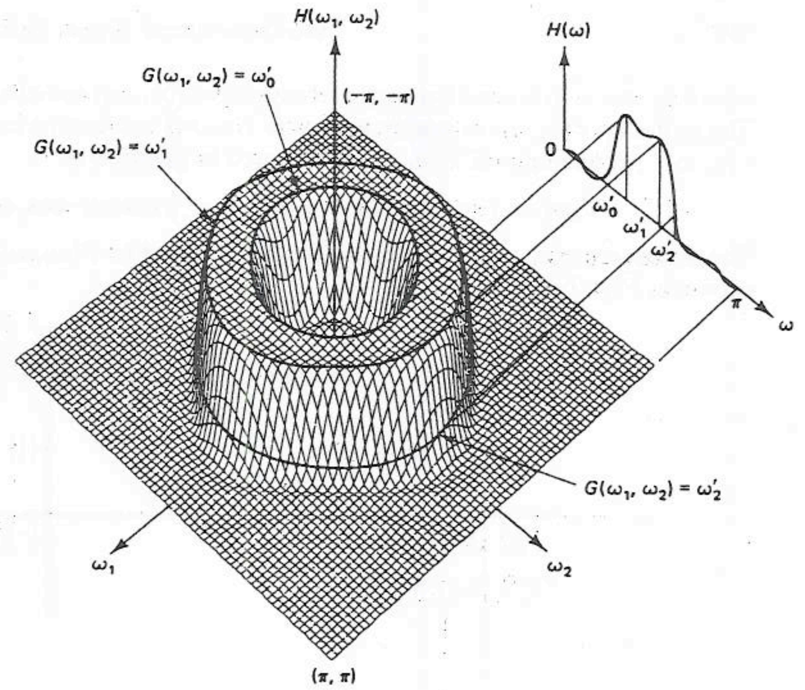


Figure 7.42 Illustration of the principle behind the design of a 2-D filter from a 1-D filter by frequency transformation.

$$\begin{aligned} H(\omega) &= \sum_{n=-N}^{N} h(n) \cdot e^{-j\omega n} = h(0) + \sum 2 \cdot h(n) \cdot \cos \omega n \\ &= \sum_{n=0}^{N} a(n) \cdot \cos \omega n = \sum_{n=0}^{N} b(n) \cdot (\cos \omega)^n \end{aligned} \tag{7.81}$$

In Eq. (7.81), the sequence $b(n)$ is not the same as $h(n)$, but it can be obtained simply from $h(n)$. The 2-D frequency response $H(\omega_1, \omega_2)$ is then obtained by

$$H(\omega_1, \omega_2) = H(\omega)\big|_{\cos \omega = T(\omega_1, \omega_2)} = \sum_{n=0}^{N} b(n) \cdot [T(\omega_1, \omega_2)]^n \tag{7.82}$$

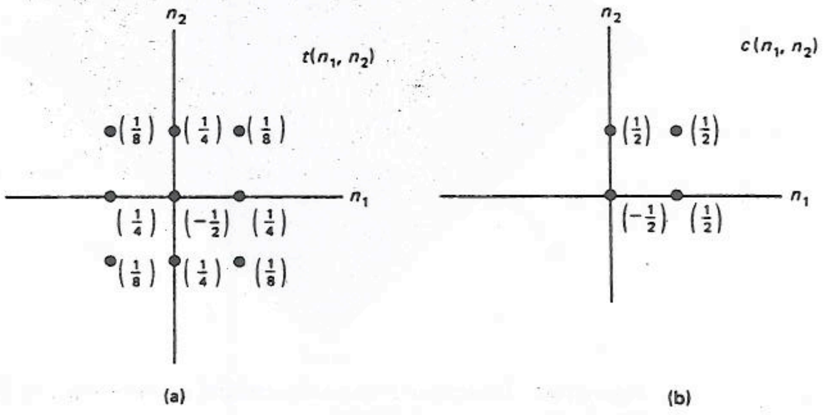
where $T(\omega_1, \omega_2)$ is the Fourier transform of a finite-extent sequence symmetric with respect to the origin, so that $T(\omega_1, \omega_2)$ can be expressed as

$$\begin{aligned} T(\omega_1, \omega_2) &= \sum\sum_{(n_1, n_2) \in R_T} t(n_1, n_2) \cdot e^{-j\omega_1 n_1} \cdot e^{-j\omega_2 n_2} \\ &= \sum\sum_{(n_1, n_2) \in R_C} c(n_1, n_2) \cdot \cos \omega_1 n_1 \cdot \cos \omega_2 n_2 \end{aligned} \tag{7.83}$$

where R_T and R_C represent the region of support of $t(n_1, n_2)$ and $c(n_1, n_2)$, respectively. The sequence $c(n_1, n_2)$ is simply related to $t(n_1, n_2)$ and can be easily obtained from $t(n_1, n_2)$. An example of $T(\omega_1, \omega_2)$ often used in practice is

$$T(\omega_1, \omega_2) = \tfrac{1}{2}\cos\omega_1 + \tfrac{1}{2}\cos\omega_2 + \tfrac{1}{2}\cos\omega_1 \cdot \cos\omega_2 - \tfrac{1}{2} \tag{7.84}$$

The sequences $t(n_1, n_2)$ and $c(n_1, n_2)$ that correspond to $T(\omega_1, \omega_2)$ in Eq. (7.84) are shown in Fig. 7.43.



n_2
$t(n_1, n_2)$
$(\frac{1}{8})$ $(\frac{1}{4})$ $(\frac{1}{8})$
n_1
$(\frac{1}{4})$ $(-\frac{1}{2})$ $(\frac{1}{4})$
$(\frac{1}{8})$ $(\frac{1}{4})$ $(\frac{1}{8})$
(a)
n_2
$c(n_1, n_2)$
$(\frac{1}{2})$ $(\frac{1}{2})$
n_1
$(-\frac{1}{2})$ $(\frac{1}{2})$
(b)

Figure 7.43 (a) A transformation sequence $t(n_1, n_2)$ and (b) the corresponding sequence $c(n_1, n_2)$ in Eq. (7.83) often used in the transformation method.

From Eqs. (7.82) and (7.83), $H(\omega_1, \omega_2)$ is always real, and therefore the resulting 2-D filter is a zero-phase filter. In addition, it is an FIR filter. For example, when $t(n_1, n_2)$ has a region of support of size $(2M_1 + 1) \times (2M_2 + 1)$ and the length of $h(n)$ is $2N + 1$, the resulting 2-D filter $H(\omega_1, \omega_2)$ is a finite-extent sequence of size $(2NM_1 + 1) \times (2NM_2 + 1)$. If $N = 10$ and $M_1 = M_2 = 1$, the region of support of $t(n_1, n_2)$ is 3×3 and the region of support of $h(n_1, n_2)$ is 21×21. In this example, the 2-D filter obtained has a large region of support for a short 1-D filter and $t(n_1, n_2)$ with a small region of support. This is typically the case.

By choosing $T(\omega_1, \omega_2)$ in Eq. (7.83) properly, we can obtain many different sets of contours that can be used for the 2-D filter design. For the transformation function $T(\omega_1, \omega_2)$ in Eq. (7.84), the set of contours obtained by $\cos\omega = T(\omega_1, \omega_2)$ for $\omega = 0, \frac{1}{10}\pi, \frac{2}{10}\pi, \ldots, \pi$ are shown in Fig. 7.44. This can be used to design many different 2-D FIR filters. From a lowpass 1-D filter of size 21 points, whose $H(\omega)$ is shown in Fig. 7.45(a), we can obtain a 2-D filter whose frequency response is shown in Fig. 7.45(b). If we begin with a 1-D highpass or bandpass filter, the resulting 2-D filter based on Eq. (7.84) would be a highpass or bandpass filter. Additional transformations and examples of filters designed by the transformation method can be found in [3].

Even though the transformation method is somewhat more complex conceptually than the window method or the frequency sampling method, its performance appears to be better than either of the two methods. In a certain restricted set of cases, the filter designed by the transformation method has been shown to be optimal. Since practical procedures do not exist for the design of optimal filters, the transformation method is one to consider in sophisticated applications that require high performance.

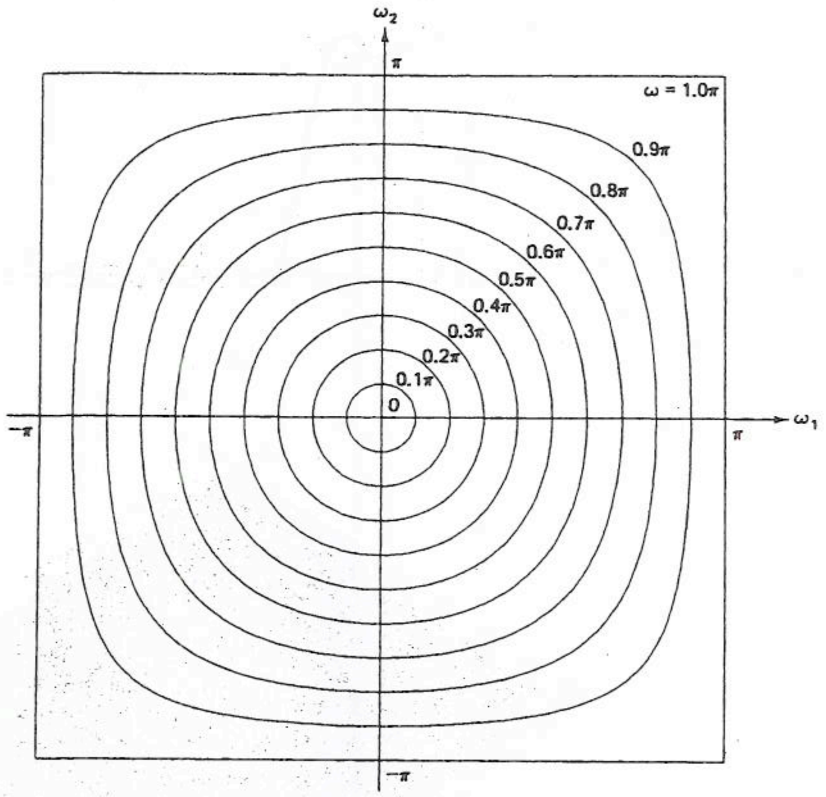


Figure 7.44 The contours obtained by $\cos\omega = T(\omega_1, \omega_2)$ for $\omega = 0, \pi/10, \ldots, \pi$ for $T(\omega_1, \omega_2)$ given by Eq. (7.84).

7.7.4 Implementation of FIR Filters

In implementing a filter, the object is to realize a discrete space system with a specified unit sample response or transfer function. The simplest method of implementing an FIR filter is to use the convolution sum. Let $x(n_1, n_2)$ and $y(n_1, n_2)$ denote the input and output of the filter. Then $y(n_1, n_2)$ is related to $x(n_1, n_2)$ by

$$y(n_1, n_2) = \sum_{(k_1, k_2) \in R_h}\sum h(k_1, k_2) \cdot x(n_1 - k_1, n_2 - k_2) \tag{7.85}$$

where R_h is the region of support of $h(n_1, n_2)$. From Eq. (7.85), the number of arithmetic operations required for each output point is about N multiplications and N additions, where N is the number of nonzero amplitudes in $h(n_1, n_2)$. As in the 1-D case, the realization can be improved by exploiting the symmetry of $h(n_1, n_2)$. Since $h(n_1, n_2) = h(-n_1, -n_2)$, by rewriting Eq. (7.85) and combining the two terms that

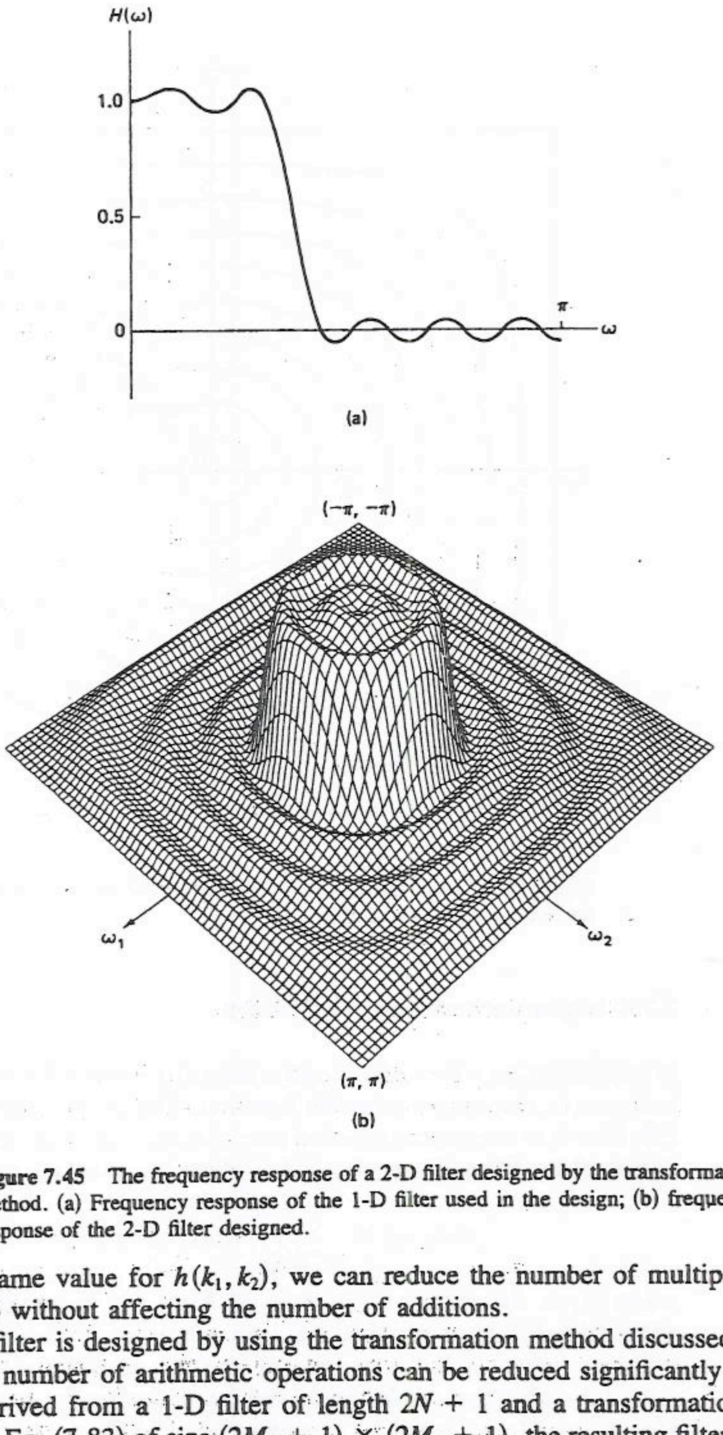


Figure 7.45 The frequency response of a 2-D filter designed by the transformation method. (a) Frequency response of the 1-D filter used in the design; (b) frequency response of the 2-D filter designed.

have the same value for $h(k_1, k_2)$, we can reduce the number of multiplications by about 50% without affecting the number of additions.

If a filter is designed by using the transformation method discussed in Section 7.7.3, the number of arithmetic operations can be reduced significantly. If the 2-D filter is derived from a 1-D filter of length $2N + 1$ and a transformation sequence $t(n_1, n_2)$ in Eq. (7.83) of size $(2M_1 + 1) \times (2M_2 + 1)$, the resulting filter $h(n_1, n_2)$ is of size $(2M_1 \cdot N + 1) \times (2M_2 \cdot N + 1)$. If the filter is implemented by direct convolution, exploiting the property that $h(n_1, n_2) = h(-n_1, -n_2)$, then the number of arithmetic operations per output sample is around $[(2M_1 \cdot N + 1)(2M_2 \cdot N + 1)]/2$ multiplications and $(2M_1 \cdot N + 1) \times (2M_2 \cdot N + 1)$ additions, which are proportional to N^2. To achieve additional computational savings, we exploit the fact (Eq. 7.82) that $H(\omega_1, \omega_2)$ designed by the transformation method is of the following form:

$$H(\omega_1, \omega_2) = \sum_{n=0}^{N} b(n) \cdot [T(\omega_1, \omega_2)]^n \tag{7.86}$$

Equation (7.86) can be realized by the system shown in Fig. 7.46. Since $T(\omega_1, \omega_2)$ corresponds to a finite-extent sequence of size $(2M_1 + 1) \times (2M_2 + 1)$, the number of arithmetic operations per output point is about $(2M_1 + 1) \cdot (2M_2 + 1) \cdot N$ multiplications and $(2M_1 + 1) \cdot (2M_2 + 1) \cdot N$ additions, which are proportional to N. For large N, this represents considerable computational savings. When $N = 20$ and $M_1 = M_2 = 1$, direct convolution with only symmetry exploitation will require about 800 multiplications and 1600 additions per output point, while the realization by Fig. 7.46 will involve 180 multiplications and 180 additions.

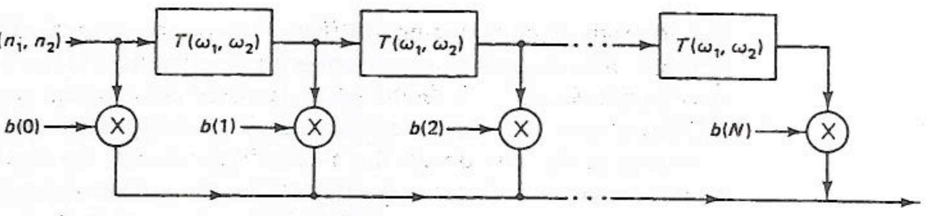


Figure 7.46 One implementation of a 2-D filter designed by the transformation method.

Any FIR filter can also be implemented by using an FFT algorithm. As we discussed in Section 7.6.1, the overlap-add method can be used to perform the filtering operation. In typical cases, this method reduces the number of arithmetic operations by a factor of 5–10 compared with realization by direct convolution.

7.8 INFINITE IMPULSE RESPONSE DIGITAL FILTERS

An infinite impulse response (IIR) filter has a unit sample response that is of infinite extent. As a result, an IIR filter differs from an FIR filter in some major ways.

An IIR filter with an arbitrary unit sample response $h(n_1, n_2)$ cannot be realized because computing each output sample may require an infinite number of arithmetic operations. As a result, in addition to requiring $h(n_1, n_2)$ to be real and stable, we require $h(n_1, n_2)$ to have a rational z-transform corresponding to a computational procedure with a wedge support output mask so that it is recursively computable. Specifically, we require $H(z_1, z_2)$, the z-transform of $h(n_1, n_2)$, to be a rational function of the following form:

$$H(z_1, z_2) = \frac{\sum\sum_{(k_1,k_2)\in R_B} b(k_1, k_2) \cdot z_1^{-k_1} \cdot z_2^{-k_2}}{\sum\sum_{(k_1,k_2)\in R_A} a(k_1, k_2) \cdot z_1^{-k_1} \cdot z_2^{-k_2}} \tag{7.87}$$

In addition, we require $h(n_1, n_2)$ to be a wedge support sequence. As we discussed in Section 7.4.5, a wedge support $h(n_1, n_2)$ with a rational z-transform can be realized by an LCCDE with proper boundary conditions.

One major difference between IIR and FIR filters is stability. An FIR filter is always stable as along as $h(n_1, n_2)$ is bounded (finite) for all (n_1, n_2), so stability is not an issue in design or implementation. With an IIR filter, however, ensuring stability is a major task. This imposes considerable restrictions on the design and implementation of IIR filters.

Zero phase is very easy to achieve for an FIR filter, and we discussed only zero-phase filters in Section 7.7. With an IIR filter, however, controlling the phase characteristics is very difficult. As a result, only the magnitude response is typically specified when an IIR filter is designed. The phase characteristic of the resulting filter is then regarded as acceptable phase. This lack of control over the phase characteristics also limits the usefulness of IIR digital filters.

7.8.1 Design of IIR Filters

The problem of designing an IIR filter is determining the coefficients of the system function. The magnitude specification that can be used is the tolerance scheme discussed in Section 7.7. In the 1-D case, there are two standard approaches to designing IIR filters. One is to design the filter from an analog filter system function, and the other is to design directly. In the 1-D IIR filter design, the first approach is typically much simpler and much more useful than the second approach. Using an elliptic analog filter system function and the bilinear transformation method, for example, optimal IIR lowpass, highpass, bandpass, and bandstop filters can be designed by following a finite fixed set of steps. Unfortunately, this approach is not useful in the 2-D case. In the 1-D case we exploit the availability of many simple methods to design 1-D analog filters that meet a given design specification. Simple methods do not exist in the design of 2-D analog filters.

In the second, direct method, an ideal unit sample response $h_d(n_1, n_2)$ or ideal magnitude response $|H_d(\omega_1, \omega_2)|$ is assumed known, and the coefficients of $H(z_1, z_2)$ are estimated so that $h(n_1, n_2)$ is closest to $h_d(n_1, n_2)$ or $|H(\omega_1, \omega_2)|$ is closest to $|H_d(\omega_1, \omega_2)|$ in some sense. The error criterion typically used is given by

$$\text{Error} = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} |h(n_1, n_2) - h_d(n_1, n_2)|^2 \tag{7.88}$$

or

$$\text{Error} = \int_{\omega_1=-\pi}^{\pi} \int_{\omega_2=-\pi}^{\pi} W(\omega_1, \omega_2)(|H(\omega_1, \omega_2)| - |H_d(\omega_1, \omega_2)|)^2 \cdot d\omega_1 \cdot d\omega_2 \tag{7.89}$$

where $W(\omega_1, \omega_2)$ is some weighting function. Minimization of Eq. (7.88) or Eq. (7.89) with respect to the coefficients of $H(z_1, z_2)$ is a highly nonlinear problem. To linearize the problem, many reasonable but ad hoc procedures have been considered. To illustrate the style in which these procedures were developed, we discuss one in particular.

Suppose $h_d(n_1, n_2)$ is given, and we wish to estimate the coefficients of $H(z_1, z_2)$ with assumed input and ouput mask shapes so that $h(n_1, n_2)$ is closest to $h_d(n_1, n_2)$. By considering the region of support of $h_d(n_1, n_2)$, the output and input mask shapes are chosen so that the resulting filter will have the same or a similar region of support. If $h_d(n_1, n_2)$ is a first-quadrant sequence, for instance, the output and input mask shapes shown in Fig. 7.47 will generate a filter with a first-quadrant support $h(n_1, n_2)$. The larger the size of the output and input masks, the better the resulting filter will be in general. The choice of the output and input mask shapes and sizes determines R_A and R_B in Eq. (7.87).

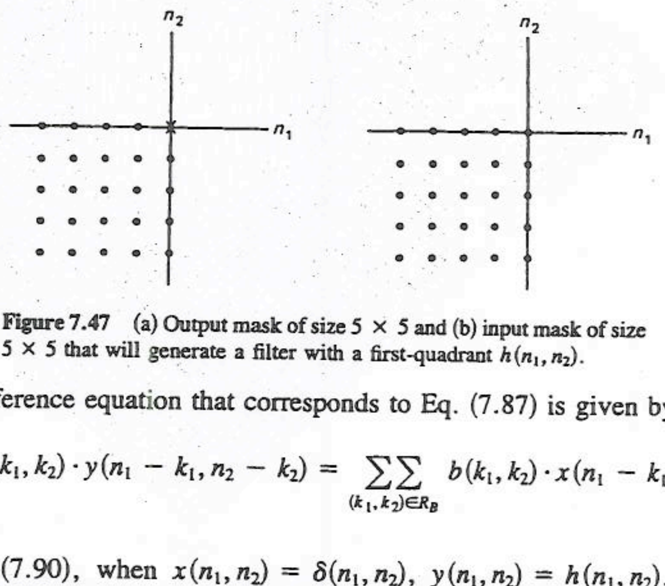


Figure 7.47 (a) Output mask of size 5×5 and (b) input mask of size 5×5 that will generate a filter with a first-quadrant $h(n_1, n_2)$.

The difference equation that corresponds to Eq. (7.87) is given by

$$\sum\sum_{(k_1,k_2)\in R_A} a(k_1,k_2)\cdot y(n_1-k_1,n_2-k_2) = \sum\sum_{(k_1,k_2)\in R_B} b(k_1,k_2)\cdot x(n_1-k_1,n_2-k_2) \tag{7.90}$$

In Eq. (7.90), when $x(n_1, n_2) = \delta(n_1, n_2)$, $y(n_1, n_2) = h(n_1, n_2)$. Therefore, from Eq. (7.90),

$$\sum\sum_{(k_1,k_2)\in R_A} a(k_1,k_2)\cdot h(n_1-k_1,n_2-k_2) = \sum\sum_{(k_1,k_2)\in R_B} b(k_1,k_2)\cdot \delta(n_1-k_1,n_2-k_2) \tag{7.91}$$

If we replace $h(n_1, n_2)$ with $h_d(n_1, n_2)$ in Eq. (7.91), we cannot expect the left-hand expression to equal the right-hand expression. Since we wish to have $h(n_1, n_2)$ as close as possible to $h_d(n_1, n_2)$, a reasonable error criterion is

$$\text{Error} = \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty}\Bigg[\sum\sum_{(k_1,k_2)\in R_A} a(k_1,k_2)\cdot h_d(n_1-k_1,n_2-k_2) - \sum\sum_{(k_1,k_2)\in R_B} b(k_1,k_2)\cdot\delta(n_1-k_1,n_2-k_2)\Bigg]^2 \tag{7.92}$$

Since the error in Eq. (7.92) is in the quadratic form of the unknown coefficients $a(n_1, n_2)$ and $b(n_1, n_2)$, minimization of error in Eq. (7.92) with respect to $a(n_1, n_2)$ and $b(n_1, n_2)$ requires solving a set of linear equations. An example of a filter designed by this method is shown in Fig. 7.48. The ideal unit sample response used is the circularly

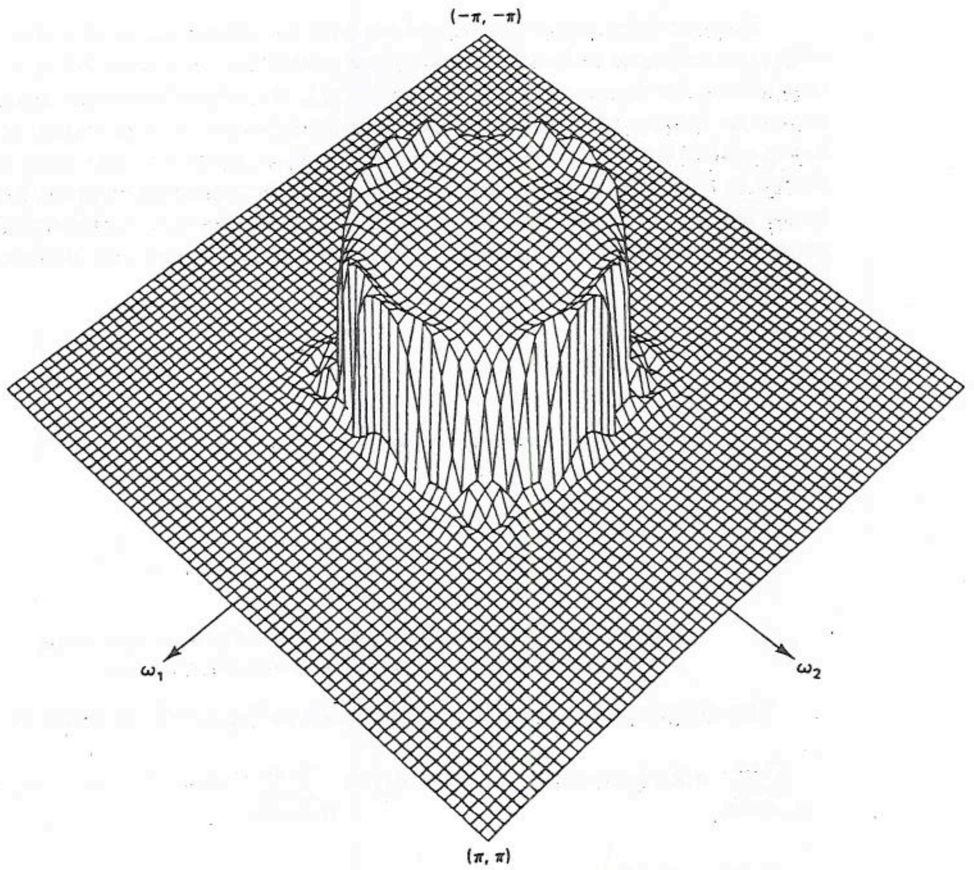


Figure 7.48 The frequency response of a 2-D zero-phase IIR filter designed.

symmetric ideal lowpass filter given by

$$h_d(n_1, n_2) = \frac{\omega_c}{2\pi\sqrt{n_1^2 + n_2^2}} \cdot J_1(\omega_c \cdot \sqrt{n_1^2 + n_2^2}) \tag{7.93}$$

To design a zero-phase filter $H(z_1, z_2)$, four one-quadrant filters $H_1(z_1, z_2)$, $H_2(z_1, z_2)$, $H_3(z_1, z_2)$, and $H_4(z_1, z_2)$ were designed. The first-quadrant filter $H_1(z_1, z_2)$ was obtained by using the output and input masks shown in Fig. 7.47 and the design method just discussed. The unit sample response used in the design of $H_1(z_1, z_2)$ is given by

$$h_d^1(n_1, n_2) = w'(n_1, n_2) \cdot h_d(n_1, n_2) = \begin{cases} h_d(n_1, n_2), & n_1, n_2 \geq 1 \\ \frac{1}{2}h_d(n_1, n_2), & n_1 = 0,\ n_2 \geq 1 \\ \frac{1}{2}h_d(n_1, n_2), & n_1 \geq 1,\ n_2 = 0 \\ \frac{1}{4}h_d(n_1, n_2), & n_1 = 0,\ n_2 = 0 \end{cases} \tag{7.94}$$

The window sequence $w'(n_1, n_2)$ used in Eq. (7.94) is shown in Fig. 7.49. The other three filters, $H_2(z_1, z_2)$, $H_3(z_1, z_2)$, and $H_4(z_1, z_2)$, were derived from $H_1(z_1, z_2)$. Specifically, the unit sample responses used in the design of $H_2(z_1, z_2)$, $H_3(z_1, z_2)$, and

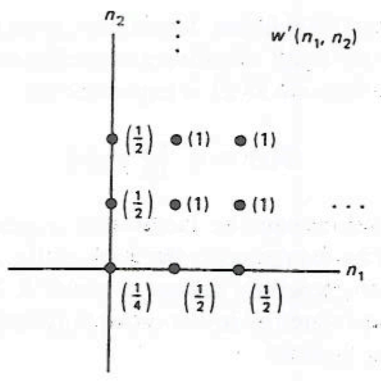


Figure 7.49 The window sequence $w'(n_1, n_2)$ in Eq. (7.94) used to design the filter in Fig. 7.48.

$H_4(z_1, z_2)$ are given by

$$h_d^2(n_1, n_2) = w'(-n_1, n_2) \cdot h_d(n_1, n_2) \tag{7.95a}$$

$$h_d^3(n_1, n_2) = w'(-n_1, -n_2) \cdot h_d(n_1, n_2) \tag{7.95b}$$

$$h_d^4(n_1, n_2) = w'(n_1, -n_2) \cdot h_d(n_1, n_2) \tag{7.95c}$$

Since $h_d^1(n_1, n_2) = h_d^2(-n_1, n_2) = h_d^3(-n_1, -n_2) = h_d^4(n_1, -n_2)$, we can obtain $H_2(z_1, z_2)$, $H_3(z_1, z_2)$, and $H_4(z_1, z_2)$ from $H_1(z_1, z_2)$ by

$$H_2(z_1, z_2) = H_1(z_1^{-1}, z_2) \tag{7.96a}$$

$$H_3(z_1, z_2) = H_1(z_1^{-1}, z_2^{-1}) \tag{7.96b}$$

$$H_4(z_1, z_2) = H_1(z_1, z_2^{-1}) \tag{7.96c}$$

Since $h_d(n_1, n_2) = h_d^1(n_1, n_2) + h_d^2(n_1, n_2) + h_d^3(n_1, n_2) + h_d^4(n_1, n_2)$, we can obtain the resulting filter $H(z_1, z_2)$ by

$$H(z_1, z_2) = H_1(z_1, z_2) + H_2(z_1, z_2) + H_3(z_1, z_2) + H_4(z_1, z_2) \tag{7.97}$$

The filter can be implemented, therefore, by a parallel combination of four recursively computable computational procedures.

In addition to the IIR filter design method discussed above, there are many variations that require solving only sets of linear equations. All these methods, of course, are not optimal procedures, and precise control over frequency-domain parameters is not possible.

7.8.2 Implementation of IIR Filters

In the implementation of 1-D IIR filters, the standard methods are direct forms, cascade forms, and parallel forms. In the implementation of 2-D IIR filters, the only method that can be used for any recursively computable rational system function is the direct-form method. In the direct-form method, a difference equation is first obtained from the system function, and the difference equation is used to recursively compute the output.

In the realization of 1-D IIR filters, the cascade form is probably the most often used because of its relatively small sensitivity to coefficient quantization. In the 1-D cascade form, the system function $H(z)$ is expressed as

$$H(z) = A \cdot \prod_k H_k(z) \tag{7.98}$$

Since a 1-D polynomial can always be factored as a product of lower-order polynomials, $H(z)$ can always be expressed in the form of Eq. (7.98). In the case of 2-D IIR filters, the cascade form generally cannot be used. A 2-D polynomial cannot, in general, be factored as a product of lower-order polynomials, and $H(z_1, z_2)$ cannot generally be written in the form of

$$H(z_1, z_2) = A \cdot \prod H_k(z_1, z_2) \tag{7.99}$$

To use a cascade form in the realization of a 2-D IIR filter, therefore, the form in Eq. (7.99) should be used explicitly in the design step.

In the 1-D parallel form, the system function $H(z)$ is expressed as

$$H(z) = \sum_k H_k(z) \tag{7.100}$$

The parallel form also requires the factorization of the denominator polynomial and therefore cannot be used as a general procedure for realizing a 2-D IIR filter. Like the cascade form, the parallel form can be used when the form in Eq. (7.100) is used explicitly in the design step. In the IIR filter design example in the previous section the form of Eq. (7.100) was used explicitly in the filter design and therefore the parallel form could be used for its implementation. The parallel form is useful when a zero-phase IIR filter is desired, as in the design example in the previous section.

7.8.3 Comparison of FIR and IIR Filters

FIR filters have many advantages over IIR filters. Stability is not an issue in FIR filter design or implementation. For IIR filters, however, testing the filter stability and stabilizing an unstable filter without significantly affecting the magnitude response is a very big task. Zero phase is extremely easy to achieve for FIR filters. Designing zero-phase IIR filters is possible, but is more involved than designing zero-phase FIR filters. In addition, design methods are simpler for FIR filters than for IIR filters.

The main advantage of an IIR filter over an FIR filter is the reduction in the number of arithmetic operations when implemented in direct form. To meet the same magnitude specification, an FIR filter typically requires more arithmetic operations per output sample than an IIR filter. If an FIR filter is implemented exploiting the computational efficiency of FFT algorithms, this advantage of an IIR filter often disappears.

Because of the overwhelming advantage of FIR filters over IIR filters, FIR filters are much more common in practice.

7.9 APPLICATIONS

The theories discussed in the previous sections in this chapter can be applied to a number of practical problems, such as images, radar signals, and geophysical data. In this section, we show a few application examples derived from image processing problems. Our objective is not to give a comprehensive treatment of the image processing field, but just to show a few examples where digital signal processing techniques have been successfully applied. Since most of the signals that arise in practice are analog, we first briefly discuss issues related to digital processing of analog signals.

7.9.1 Digital Processing of Analog Signals

The issues that arise in digital processing of analog signals are essentially the same for the 1-D and the 2-D case, and therefore we simply summarize the 2-D results.

To differentiate an analog signal from a sequence, we denote the analog signal by $x_a(t_1, t_2)$. The continuous space Fourier transform of $x_a(t_1, t_2)$, $X_a(\Omega_1, \Omega_2)$, is related to $x_a(t_1, t_2)$ by

$$X_a(\Omega_1, \Omega_2) = \int_{t_1=-\infty}^{\infty} \int_{t_2=-\infty}^{\infty} x_a(t_1, t_2) \cdot e^{-j\Omega_1 t_1} \cdot e^{-j\Omega_2 t_2} \cdot dt_1 \cdot dt_2 \tag{7.101}$$

$$x_a(t_1, t_2) = \frac{1}{(2\pi)^2} \int_{\Omega_1=-\infty}^{\infty} \int_{\Omega_2=-\infty}^{\infty} X_a(\Omega_1, \Omega_2) \cdot e^{j\Omega_1 t_1} \cdot e^{j\Omega_2 t_2} \cdot d\Omega_1 \cdot d\Omega_2 \tag{7.102}$$

Suppose we obtain a discrete space signal $x(n_1, n_2)$ by sampling an analog signal $x_a(t_1, t_2)$ with sampling period (T_1, T_2) as follows:

$$x(n_1, n_2) = x_a(t_1, t_2)\big|_{t_1=n_1T_1,\, t_2=n_2T_2} \tag{7.103}$$

Equation (7.103) represents the input-output relationship of an ideal analog-to-digital (A/D) converter. The relation between $X(\omega_1, \omega_2)$, the discrete space Fourier transform of $x(n_1, n_2)$, and $X_a(\Omega_1, \Omega_2)$, the continuous space Fourier transform of $x_a(t_1, t_2)$, is

$$X(\omega_1, \omega_2) = \frac{1}{T_1 \cdot T_2} \cdot \sum_{r_1=-\infty}^{\infty} \sum_{r_2=-\infty}^{\infty} X_a\left(\frac{\omega_1 - 2\pi r_1}{T_1}, \frac{\omega_2 - 2\pi r_2}{T_2}\right) \tag{7.104}$$

Examples of $X_a(\Omega_1, \Omega_2)$ and $X(\omega_1, \omega_2)$ are shown in Fig. 7.50 for the case $1/T_1 > \Omega_c/\pi$ and $1/T_2 > \Omega_c'/\pi$, where Ω_c and Ω_c' are the cutoff frequencies of $X_a(\Omega_1, \Omega_2)$, as shown. From the figure, when $1/T_1 > \Omega_c/\pi$ and $1/T_2 > \Omega_c'/\pi$, $x_a(t_1, t_2)$ can be recovered from $x(n_1, n_2)$. This is the 2-D sampling theorem, which is a straightforward extension of the 1-D results.

An ideal digital-to-analog (D/A) converter recovers $x_a(t_1, t_2)$ from $x(n_1, n_2)$ when the sampling frequencies $1/T_1$ and $1/T_2$ are sufficiently high to satisfy the sampling

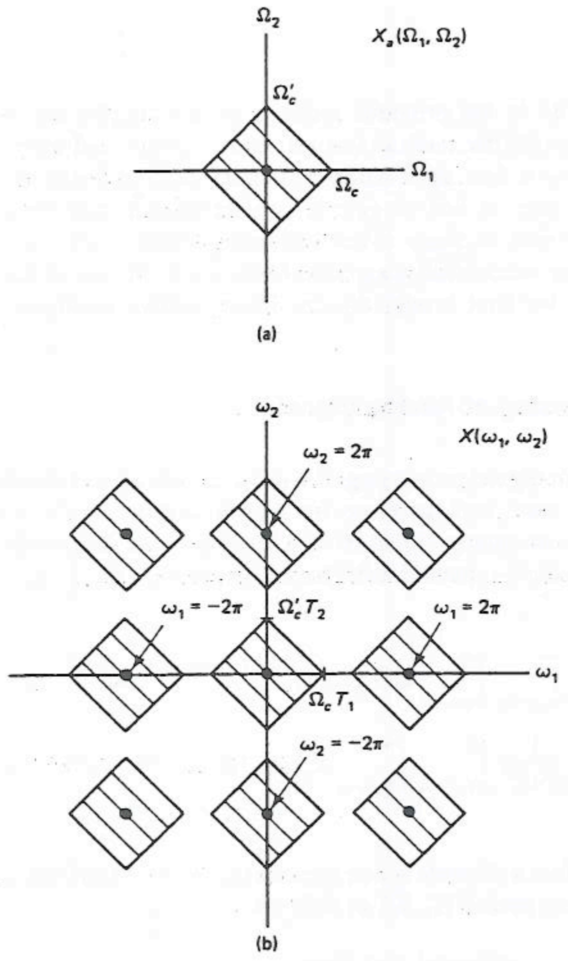


Figure 7.50 An example illustrating the relationship between $X_a(\Omega_1, \Omega_2)$ and $X(\omega_1, \omega_2)$ given by Eq. (7.104). (a) $X_a(\Omega_1, \Omega_2)$; (b) $X(\omega_1, \omega_2)$.

theorem. The output of the ideal D/A converter, $y_a(t_1, t_2)$, is given by

$$y_a(t_1, t_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1, n_2) \cdot \frac{\sin \frac{\pi}{T_1}(t_1 - n_1 \cdot T_1)}{\frac{\pi}{T_1}(t_1 - n_1 \cdot T_1)} \cdot \frac{\sin \frac{\pi}{T_2}(t_2 - n_2 \cdot T_2)}{\frac{\pi}{T_2}(t_2 - n_2 \cdot T_2)} \tag{7.105}$$

The function $y_a(t_1, t_2)$ is identical to $x_a(t_1, t_2)$ when the sampling frequencies used in the ideal A/D converter are sufficiently high. Otherwise, $y_a(t_1, t_2)$ is an aliased version of $x_a(t_1, t_2)$. Equation (7.105) is a straightforward extension of the 1-D results.

An analog signal can often be processed by digital signal processing techniques using the A/D and D/A converters just discussed. Digital processing of analog signals can, in general, be represented by the system in Fig. 7.51. The analog lowpass filter limits the bandwidth of the analog filter to reduce the effect of aliasing.

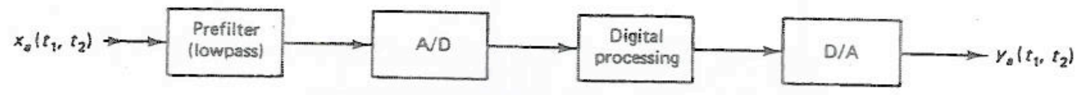


Figure 7.51 Digital processing of analog signals.

7.9.2 Examples in Image Processing Applications

An important application area of 2-D signal processing theories is image processing, which in recent years has received considerable attention. This is due in part to significant advances in hardware technology that allow sophisticated image processing algorithms to be implemented in real time and in part to a large number of applications of image processing in such diverse areas as medicine, communications, consumer electronics, defense, law enforcement, robotics, geophysics, and agriculture. Image processing can be classified broadly into four areas: image restoration, enhancement, coding, and understanding. In this section, we illustrate one example in each area.

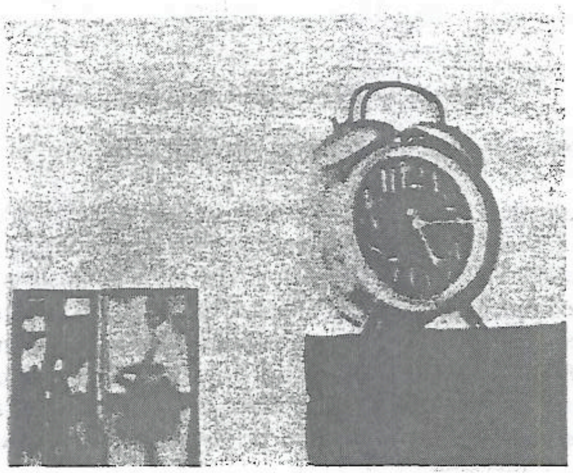
Image restoration. In image restoration, an image has been degraded in some manner and the objective is to reduce or eliminate the effect of degradation. Typical degradations that occur in practice include image blurring, additive random noise, quantization noise, multiplicative noise, and geometric distortion. Suppose an image $f(n_1, n_2)$ is degraded by additive random noise. Then the degraded image $g(n_1, n_2)$ can be expressed as

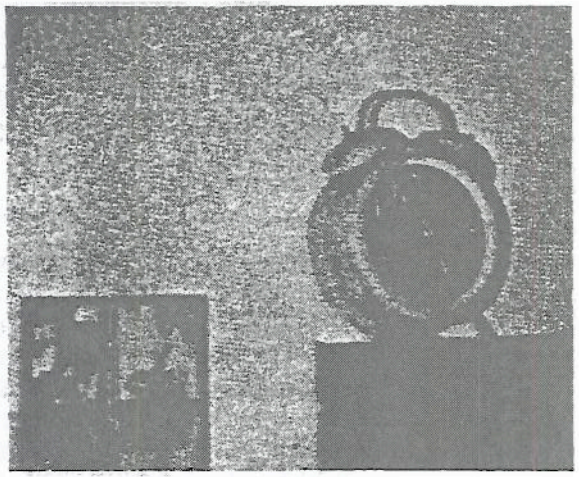
$$g(n_1, n_2) = f(n_1, n_2) + w(n_1, n_2) \tag{7.106}$$

where $w(n_1, n_2)$ is a random background noise. An example of an image degraded by white noise is shown in Fig. 7.52. Part (a) shows an undegraded original image of 256×256 pixels, and part (b) shows the degraded image.

An image processed by a signal processing algorithm to reduce additive noise in the image in Fig. 7.52(b) is shown in Fig. 7.52(c). The degraded image is filtered by a space-variant FIR filter. A new filter was designed at each pixel based on the amount of image detail in the neighborhood of the pixel to be processed. If there is a fair amount of detail, such as near edges, then little lowpass filtering is performed since details generally correspond to high-frequency components and therefore a high level of lowpass filtering can reduce the image details. In addition, the same amount of noise in high-detail image areas is less visible than in low-detail image areas. In low-detail image areas such as uniform background areas, a large amount of lowpass filtering is performed. Low details in images generally correspond to low-frequency components, and therefore the signal is not significantly degraded by a large amount of lowpass filtering. The background noise, on the other hand, is significantly reduced by a large amount of lowpass filtering.

Image enhancement. Image enhancement is the processing of an image to improve its visual appearance to a human viewer or to enhance the performance of another image processing system. Methods and objectives vary with the application. When images are enhanced for human viewers, as in television, the objective may be to improve perceptual aspects: image quality, intelligibility, or visual appearance. In


(a)


(b)

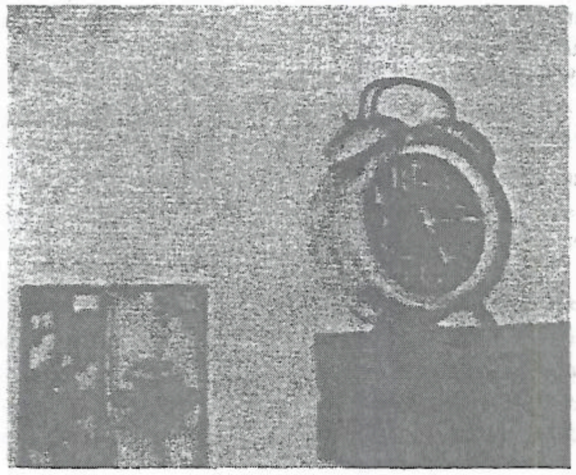

(c)

Figure 7.52 (a) An original image of 256×256 pixels; (b) the image in part (a) degraded by additive white noise; (c) a restored image.

applications such as object identification by machine, an image may be preprocessed to aid machine performance. Because the objective of image enhancement is heavily dependent on the application context and the criteria for enhancement are often subjective or too complex to be easily converted to useful objective measures, image enhancement algorithms tend to be simple, qualitative, and ad hoc. Image enhancement is closely related to image restoration. When an image is degraded, restoration of the original image often results in enhancement. There are, however, some important differences. In image restoration, an ideal image has been degraded and the objective is to make the processed image resemble the original as much as possible. In image enhancement, the objective is to make the processed image better in some sense than the unprocessed image. To understand this difference, note that the original, undegraded image cannot be further restored but can be enhanced by increasing sharpness.

The visual appearance of an image can often be enhanced significantly by proper manipulation of the contrast and the overall dynamic range of the image. An example of this is illustrated in Fig. 7.53. Part (a) shows an image that was taken from an airplane. Because of the varying amounts of cloud cover, the details on the ground are not very visible. Part (b) shows an image obtained by processing the image in part (a). In the processing, different operations were performed for different regions of the image. By measuring the local average intensity in a particular region, we can estimate the approximate level of cloud cover. The high average intensity region generally corresponds to a higher level of cloud cover. In regions where cloud cover appears to be present, the image is highpass filtered to increase the contrast. In addition, the local average intensity is reduced so that the contrast increase will not be clipped because of dynamic range increase caused by the contrast increase. The amount of highpass filtering and reduction in the local average intensity is adapted to the estimated level of cloud cover. In this example, the highpass filter used is FIR.

Image coding. The objective in image coding is to represent an image with as few bits as possible, preserving a certain level of image quality and intelligibility acceptable for a given application. Image coding can be used in reducing the bandwidth of a communication channel when an image is transmitted and in reducing the amount of required storage when an image needs to be retrieved at some future time. Image coding is related to image restoration and enhancement. If we can reduce the degradation such as quantization noise that results from an image coding algorithm or to enhance the visual appearance of the reconstructed image, for example, we can reduce the number of bits required to represent the image at a given level of image quality and intelligibility.

One approach to image coding is transform coding, in which an image is first transformed into a different domain and then the transformed image is coded. For transforms such as the Fourier transform, the energy of typical images is concentrated in a small region and therefore only the transform coefficients in the small region can be coded without significant distortion of an image. An example that illustrates the performance of a transform coding method is shown in Fig. 7.54. Part (a) shows an image of 256×256 pixels where each pixel value is represented by 8 bits of uniform quantization. Part (b) shows the coded image obtained by coding only a small number

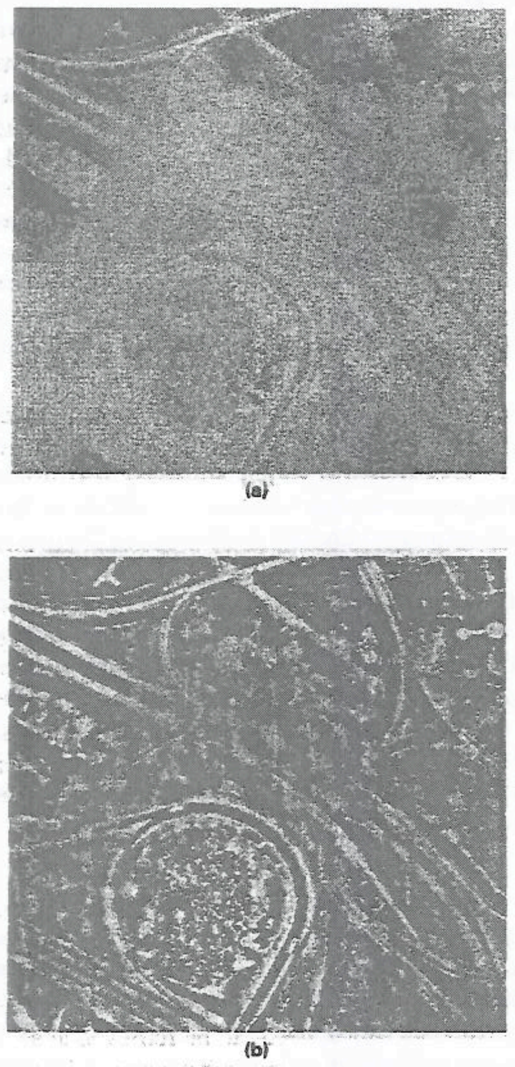
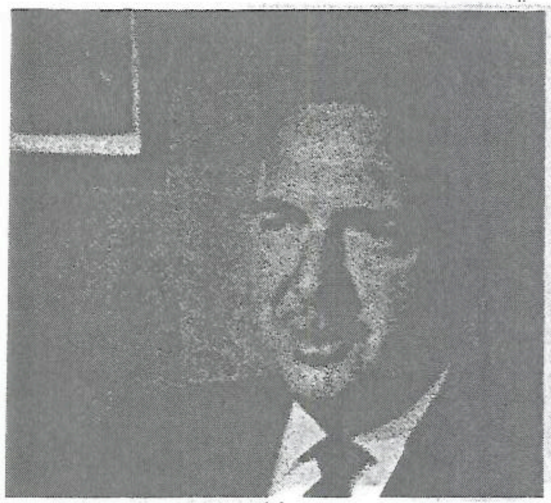


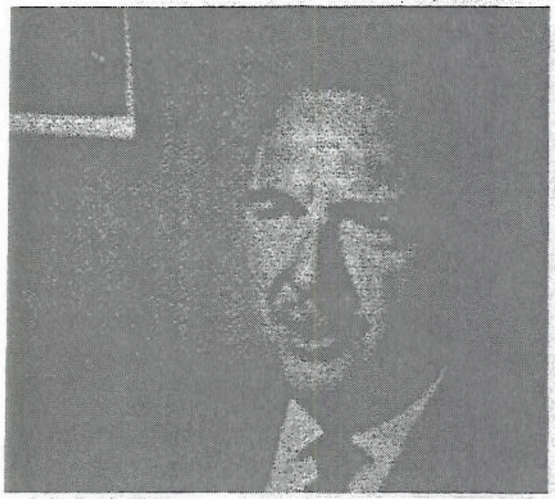
Figure 7.53 (a) An image of 256 × 256 pixels taken from an airplane; (b) the image in part (a) processed for enhancement.

of discrete cosine transform coefficients. The discrete cosine transform is closely related to the discrete Fourier transform. The number of bits used for this image is 0.7 bit/pixel. If a straightforward pulse code modulation (PCM) system that codes the intensity of the image were used, at least 2–4 bits/pixel would be necessary to obtain an image quality similar to the one in Fig. 7.54(b).

Image understanding. The objective in image understanding is to symbolically represent the contents of an image. Applications of image understanding include computer vision, robotics, and target identification. Image understanding differs from the other three areas in one major aspect. In image understanding, the input is an image, but the output is typically some symbolic representation of the contents of the image. Successful development of a system in this area generally requires both signal



(a)

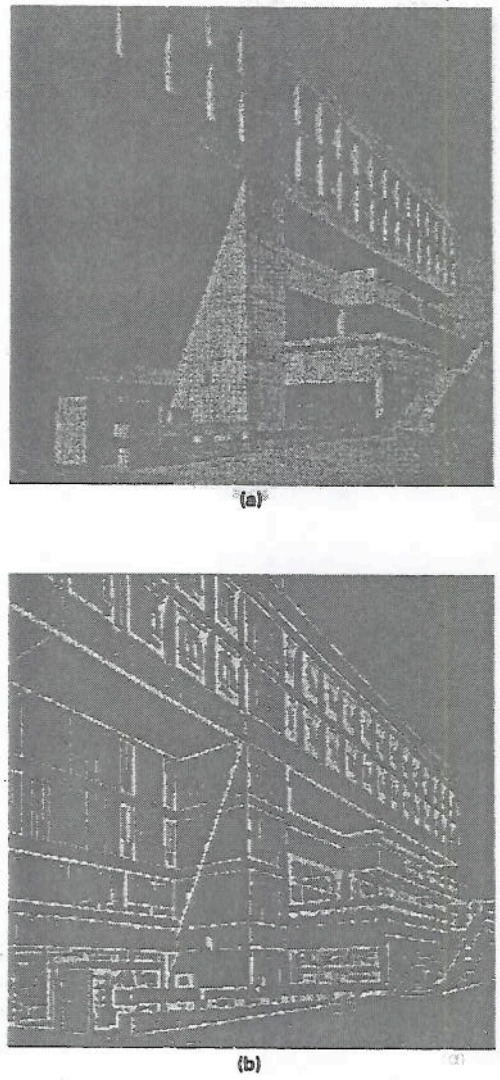


(b)

Figure 7.54 (a) An original image of 256 × 256 pixels; (b) the image in part (a) coded by a discrete cosine transform method at 0.7 bit/pixel.

processing and artificial intelligence concepts. In a typical image understanding system, signal processing is used to perform lower-level processing such as reduction of degradation and extraction of image features such as edges, and artificial intelligence is used to perform higher-level processing such as symbol manipulation and knowledge-base management.

An example of edges detected by a simple signal processing algorithm is shown in Fig. 7.55. Part (a) shows an original image of 256 × 256 pixels. Part (b) shows the image where the edges or intensity discontinuities are shown. The edges were obtained by applying a bandpass filter to the original image in part (a) to emphasize intensity discontinuities and then applying a threshold test. Regions of the bandpass filtered image are declared to be edges when the pixel values are above a certain threshold.


(a)

(b)

Figure 7.55 (a) An original image of 256×256 pixels; (b) the edge contour obtained from the image in part (a).

7.10 SUMMARY

In this chapter, we discussed the fundamentals of 2-D signal processing, including the Fourier transform, the z-transform, difference equations, discrete Fourier transform, fast Fourier transform, and design and implementation of digital filters. These are the same topics typically discussed in fundamentals of 1-D signal processing, but there are a number of differences between 1-D signal processing and 2-D signal processing, which we have attempted to show. Although our discussion in this chapter concentrated on 2-D signal processing, we note that the results can be extended in a straightforward manner to higher-dimensional signal processing. Those who wish to study 2-D signal processing in greater detail should refer to [4,5,6,7,8,9].

REFERENCES

1. J. O. Eklundh, "A Fast Computer Method for Matrix Transposing," *IEEE Trans. Computers*, Vol. 21, pp. 801–803, 1972.
2. T. W. Parks and J. H. McClellan, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Trans. Circuit Theory*, Vol. 19, pp. 189–194, 1972.
3. R. M. Mersereau, W. F. G. Mecklenbrauken, and T. F. Quatieri, Jr., "McClellan Transformations for Two-Dimensional Digital Filtering: I, Design." *IEEE Trans. Circuits and Systems*, Vol. 23, pp. 405–414, July 1976.
4. J. S. Lim, *Two-Dimensional Signal Processing and Image Processing*, to be published.
5. D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1983.
6. T. S. Huang, Ed., *Two-Dimensional Digital Signal Processing I*, in *Topics in Applied Physics*, Vol. 42, Springer-Verlag, Berlin, 1981.
7. T. S. Huang, Ed., *Two-Dimensional Digital Signal Processing II*, in *Topics in Applied Physics*, Vol. 43, Springer-Verlag, Berlin, 1981.
8. S. K. Mitra and M. P. Ekstron, Eds., *Two-Dimensional Digital Signal Processing*, Dowden, Hutchinson, and Ross, Stroudsburg, PA, 1978.
9. IEEE ASSP Society's MDSP Technical Committee, Ed., *Selected Papers in Multidimensional Digital Signal Processing*, IEEE Press, New York, 1986.

8

Some Advanced Topics in Filter Design

Hans Wilhelm Schüssler
Peter Steffen
University of Erlangen-Nuremberg

8.0 INTRODUCTION

The term *filter* most typically denotes a device having selective properties. In the ideal case, some parts of the spectrum of the incoming signal are passed without any change while other parts are suppressed completely. This process can be expressed by specifying a desired idealized transfer function $H_i(\omega)$. In the case of a lowpass filter, for example,

$$H_i(\omega) = \begin{cases} 1 & \text{in the passband} \\ 0 & \text{in the stopband} \end{cases}$$

In reality these properties cannot be achieved exactly. Fortunately, they are not necessary. A certain constant delay can always be tolerated, and some deviations from the desired behavior in the passband and stopband as well as in a transition band between both are permissible [1]. This leads to tolerance schemes for the magnitude and group delay of the system to be designed. An example of a lowpass filter is shown in Fig. 8.1. In (b) specifications for the group delay are given in the passband only.

The following remarks can be made about the filter shown in the figure:

1. The parameters ω_p, ω_s, δ_1, δ_2, and Δ as well as τ_0 depend on the particular application. In many cases of practical interest a phase distortion of the output signal is acceptable. Thus only the tolerance scheme for the magnitude in Fig. 8.1(a) has to be satisfied.

Hans Wilhelm Schüssler and Peter Steffen are with Lehrstuhl für Nachrichtentechnik, Universität Erlangen-Nürnberg, Cauerstrasse 7, 8520 Erlangen, West Germany.